# Manual do Usuário - Analisador Léxico

## Visão Geral

Este projeto implementa um analisador léxico (scanner) para uma linguagem de programação personalizada. O analisador utiliza o gerador de analisadores léxicos **Flex** para reconhecer tokens de uma linguagem que combina elementos de francês e português.

**Para a especificação formal completa da linguagem**, consulte o documento Linguagem Regular.

## Funcionalidades

O analisador reconhece os seguintes elementos:

### Palavras-chave

- `siu` - equivalente a "if" (se)
- `autre` - equivalente a "else" (senão)
- `sinon siu` - equivalente a "elif" (senão se)
- `changer` - equivalente a "switch" (escolha)
- `cas` - equivalente a "case" (caso)
- `defaut` - equivalente a "default" (padrão)
- `casser` - equivalente a "break" (quebrar)
- `continuer` - equivalente a "continue" (continuar)
- `retour` - equivalente a "return" (retornar)
- `faire` - equivalente a "do" (fazer)
- `dembele` - equivalente a "while" (enquanto)
- `mbappe` - equivalente a "for" (para)

### Operadores

- `+` - soma
- `-` - subtração
- `X` - multiplicação
- `/` - divisão

- `:` - atribuição
- `Λ` - menor que
- `V` - maior que
- `V/` - menor ou igual
- `::` - igualdade
- `ney:` - diferente
- `ney` - negação

### Delimitadores

- `;` - ponto e vírgula
- `,` - vírgula
- `(` e `)` - parênteses
- `{` e `}` - chaves

### Tipos de Dados

- **Identificadores**: começam com letra (exceto V e X que são operadores), podem conter letras, dígitos e underscore
- **Números inteiros**: sequência de dígitos
- **Números decimais**: formato `123.456`

### Comentários

- `//` - comentário de linha única
- `/* */` - comentário de múltiplas linhas

## Instalação

### Pré-requisitos

```
# Ubuntu/Debian
sudo apt-get install flex gcc

# Fedora/CentOS
sudo yum install flex gcc
```

```
# Arch Linux
sudo pacman -S flex gcc
```

### Compilação

```
# Verificar dependências
make check-deps

# Compilar o projeto
make

# Ou compilar do zero
make rebuild
```

## Uso

### Sintaxe Básica

```
./analisador <arquivo_de_entrada>
```

### Exemplos

1. **Teste básico:**

```
make test
```

2. **Arquivo personalizado:**

```
echo 'siu (x : 10) faire' > meu_teste.txt
./analisador meu_teste.txt
```

3. **Programa completo:**

```
cat > exemplo.txt << EOF
siu (contador : 0) {
    faire {
        contador : contador + 1;
```

```
    } dembele (contador Λ 10);
    retour contador;
}

mbappe (i : 0; i Λ 5; i : i + 1) {
    resultado : i X 2;
}
EOF

./analisador exemplo.txt
```

### Saída Esperada

```
LINHA 1: KEYWORD_IF -> 'siu'
LINHA 1: DELIM_ABRE_PARENTESES -> '('
LINHA 1: IDENTIFICADOR -> 'contador'
LINHA 1: OP_ATRIB -> ':'
LINHA 1: NUMERO_INT -> 0
LINHA 1: DELIM_FECHA_PARENTESES -> ')'
LINHA 1: DELIM_ABRE_CHAVES -> '{'
...
```

## Comandos Make Disponíveis

| Comando | Descrição |
|---|---|
| `make` | Compila o projeto |
| `make test` | Executa teste básico |
| `make clean` | Remove arquivos gerados |
| `make rebuild` | Recompila do zero |
| `make install` | Instala no sistema (/usr/local/bin) |
| `make check-deps` | Verifica dependências |
| `make info` | Mostra informações do projeto |
| `make help` | Mostra ajuda |

## Estrutura do Projeto

```
C/Compiler/
├── Makefile              # Script de compilação
├── src/
│   ├── analisador.l      # Código fonte do analisador
│   └── lex.yy.c          # Código C gerado pelo flex
├── tests/
│   └── teste_completo.txt # Arquivo de teste
├── docs/
│   ├── MANUAL.pdf                # Este manual
|   └── linguagem_regular.pdf   # Arquivo de teste
└── analisador                  # Executável (gerado)
```

# Resolução de Problemas

## Erro: "flex: command not found"

```
sudo apt-get install flex
```

## Erro: "gcc: command not found"

```
sudo apt-get install gcc
```

## Erro: "undefined reference to yywrap"

Certifique-se de usar a flag `-lfl` na compilação:

```
gcc lex.yy.c -lfl -o analisador
```

## Warning: "rule cannot be matched"

Alguns warnings são normais devido à ordem das regras. O analisador funciona corretamente mesmo com esses warnings.

# Desenvolvimento

## Modificando o Analisador

1. Edite o arquivo `src/analisador.l`
2. Recompile com `make rebuild`
3. Teste com `make test`

### Adicionando Novos Tokens

Para adicionar um novo token, edite a seção de regras em `src/analisador.l` :

```
"nova_palavra"  {printf("LINHA %d: NOVO_TOKEN -> '%s'\n", line_count, yyt
```

### Testando Alterações

1. Crie um arquivo de teste
2. Execute: `./analisador seu_arquivo.txt`
3. Verifique se a saída está correta

## Arquivo de Demonstração

O projeto inclui um arquivo completo de demonstração em `tests/arquivo_fonte.txt` que mostra todas as funcionalidades do analisador léxico.

### Conteúdo do Arquivo de Demonstração

O arquivo `tests/arquivo_fonte.txt` é um programa completo que demonstra:

- **Todas as palavras-chave** da linguagem
- **Todos os operadores** (aritméticos, relacionais, lógicos)
- **Números** (inteiros e decimais)
- **Strings literais** com diferentes conteúdos
- **Identificadores válidos** com diferentes padrões
- **Comentários** de linha única e múltiplas linhas
- **Estruturas de controle** completas (if, switch, for, while)
- **Erros léxicos intencionais** para demonstrar detecção de erros

### Como Usar o Arquivo de Demonstração

```
# Analisar o arquivo completo
./analisador tests/arquivo_fonte.txt
```

```
# Ver apenas os primeiros 30 tokens
./analisador tests/arquivo_fonte.txt | head -30

# Ver apenas os erros léxicos
./analisador tests/arquivo_fonte.txt | grep "ERRO"

# Contar total de tokens gerados
./analisador tests/arquivo_fonte.txt | wc -l

# Analisar tipos específicos de tokens
./analisador tests/arquivo_fonte.txt | grep "KEYWORD"
./analisador tests/arquivo_fonte.txt | grep "NUMERO"
./analisador tests/arquivo_fonte.txt | grep "STRING"
```

## Exemplo de Saída

Quando você executar `./analisador tests/arquivo_fonte.txt`, verá saídas como:

```
LINHA 10: KEYWORD_IF -> 'siu'
LINHA 10: DELIM_ABRE_PARENTESES -> '('
LINHA 10: IDENTIFICADOR -> 'idade'
LINHA 10: OP_MENOR_IGUAL -> 'V/'
LINHA 10: NUMERO_INT -> 18
LINHA 10: DELIM_FECHA_PARENTESES -> ')'
LINHA 11: STRING -> "adulto"
LINHA 117: ERRO LÉXICO! Caractere desconhecido: '@'
LINHA 118: ERRO LÉXICO! Caractere desconhecido: '#'
...
```

## Seções do Arquivo de Demonstração

1. **Palavras-chave e estruturas de controle** - if, else, switch, case
2. **Laços de repetição** - for (mbappe) e while (dembele)
3. **Operadores aritméticos** - soma, subtração, multiplicação, divisão
4. **Operadores relacionais** - comparações e igualdades
5. **Números e strings** - diferentes formatos de literais
6. **Identificadores válidos** - vários padrões aceitos
7. **Negação lógica** - operador ney
8. **Delimitadores** - parênteses, chaves, vírgulas
9. **Comentários** - linha única e múltiplas linhas
10. **Erros léxicos** - caracteres inválidos para demonstração

11. **Casos especiais** - situações de borda
12. **Programa completo** - exemplo de código real

### Validação e Teste

Use o arquivo de demonstração para:

- **Validar modificações** no analisador
- **Aprender a sintaxe** da linguagem
- **Testar novos recursos** adicionados
- **Verificar detecção de erros**
- **Demonstrar funcionalidades** para outros usuários

## Suporte

Para problemas ou dúvidas:

1. Verifique se todas as dependências estão instaladas ( `make check-deps` )
2. Tente recompilar do zero ( `make rebuild` )
3. Execute um teste básico ( `make test` )