# Relatório de Análise Léxica do MicroPascal

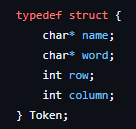
## Introdução

Este relatório descreve a implementação de um analisador léxico para a linguagem de programação MicroPascal ou μ-Pascal, utilizando a linguagem C. O analisador é responsável por processar um código fonte em Pascal e identificar os diferentes elementos lexicais, categorizando-os como palavras reservadas, identificadores, valores numéricos, operadores, símbolos, etc.

## Estruturas de Dados

**Struct Token**

A struct token é utilizada para representar as unidades léxicas identificadas no código fonte.



### Propósito

* **Name**: Armazena o tipo de token como por exemplo se é uma palavra reservada, um operador, etc.
* **Word**: Contém o texto literal correspondente ao token.
* **Row** e **Column**: Permitem rastrear a localização do token no código fonte, facilitando a identificação de erros lexicais.

## Funções

**Função: lexerAnalysis**

### 

A função lexerAnalysis é o núcleo do analisador léxico, que lê o código fonte caractere por caractere e identifica tokens.

1. Inicializa variáveis de controle, como row, column e state para gerenciar o estado atual da análise.
2. Utiliza um loop para ler o arquivo de entrada até que o final do arquivo seja alcançado.
3. Dependendo do estado atual, ignora espaços em branco e quebras de linha, identifica se o caractere é uma letra, um dígito ou um símbolo especial e chama funções auxiliares para tratar palavras reservadas, identificadores e números
4. Quando um token é identificado, a função createToken é chamada para criar um novo token que é então armazenado em uma lista.
5. Retorna um ponteiro para a lista de tokens identificados ao final do processo.

**Função: createToken**



Cria um novo token e aloca memória para ele. Seguindo o seguinte fluxo:

1. Recebe como parâmetros o tipo do token, name, seu texto, word, e suas coordenadas no código, row e column.
2. Aloca memória para um novo Token
3. Preenche os campos do token com os valores fornecidos.
4. Retorna um ponteiro para o token recém-criado.

**Função showError**



Exibe mensagens de erro quando caracteres desconhecidos ou situações anômalas são encontradas. Tendo como forma de funcionamento:

1. Recebe uma mensagem de erro e as coordenadas onde o erro ocorreu.
2. Imprime a mensagem de erro no console.
3. Pode registrar a mensagem em um arquivo de saída, se implementado.

**Função isReservedWord e isReservedType**



Ambos verificam se a palavra e o tipo de dado estão reservados a linguagem Micropascal, funcionando de maneira parecida.

1. Compara a palavra fornecida com uma lista pré-definida de palavras reservadas.
2. Retorna 1 se a palavra for reservada e 0 caso contrário.

**Função isReservedOperator**



Determina se a palavra corresponde a um operador reservado da linguagem. Compara a palavra fornecida com uma lista de operadores pré-definida e retorna 1 se for um operador ou 0 caso contrário.

**Função addWord**



Adiciona um caractere à string word e ajusta seu tamanho. Seguindo o seguinte padrão:

1. Recebe um ponteiro para a string, um ponteiro para o tamanho atual da string, e o caractere a ser adicionado.
2. Realiza a alocação de memória necessária se o tamanho da string precisar ser aumentado.
3. Adiciona o caractere à string e termina a string corretamente.

# Testes

### Testes com Programas Corretos

### Teste 1 – código:

|  |
| --- |
| program Teste1;vara: integer;begina := 5;end. |

### Saída do analisador teste 1:

|  |
| --- |
| Tokens identificados:<program, Teste1><var, var><ID, a><integer, integer><ID, a><OP\_ASS, :=><NUM, 5><begin, begin><end, end> |

### Teste 2 – código:

|  |
| --- |
| program Teste2;varx, y: integer;beginif x > y thenx := y;end. |

### Saída do analisador teste 2:

|  |
| --- |
| Tokens identificados:<program, Teste2><var, var><ID, x><ID, y><integer, integer><if, if><ID, x><OP\_GT, ><ID, y><then, then><ID, x><OP\_ASS, :=><ID, y> |

### Teste 3 – código:

|  |
| --- |
| program Teste3;vara, b: real;begina := 10.5;b := a \* 2.0;end. |

### Saída do analisador teste 3:

|  |
| --- |
| Tokens identificados:<program, Teste3><var, var><ID, a><real, real><ID, b><real, real><ID, a><OP\_ASS, :=><NUM, 10.5><ID, b><OP\_ASS, :=><ID, a><OP\_MUL, \*><NUM, 2.0><begin, begin><end, end> |

### Testes com Programas Incorretos

### Teste 1 – código:

|  |
| --- |
| program TesteErro;varz: real;beginz = 10; // Erro: operador de atribuição incorretoend. |

### Saída do analisador teste 1:

|  |
| --- |
| Erro: operador de atribuição esperado na linha 4, coluna 9. |

### Teste 2 – código:

|  |
| --- |
| program TesteErro2;vara: integer;beginif a > 10end. |

### Saída do analisador teste 2:

|  |
| --- |
| Erro: falta de 'then' na linha 5, coluna 10. |

### Teste 3 – código:

|  |
| --- |
| program TesteErro3;varx, y: integer;beginwhile x < 10 dox := x + 1y := y + 1; // Erro: falta de ponto e vírgulaend. |

### Saída do analisador teste 3:

|  |
| --- |
| Erro: falta de ponto e vírgula na linha 6, coluna 17. |

# Autômatos Infinitos Determinísticos (AFD)

# AFD – identificadores e operações com números reais e inteiros

# Diagrama Descrição gerada automaticamente

# AFD – palavras reservadas

# Diagrama Descrição gerada automaticamente

# AFD – simbolos Diagrama Descrição gerada automaticamente