Relatório Técnico - Analisador Sintático do MicroPascal

1. Introdução

Este projeto visa a implementação de um analisador sintático e léxico para uma linguagem baseada na gramática do MicroPascal. O analisador léxico é responsável por dividir o código-fonte em tokens, e o analisador sintático valida se o código segue as regras definidas pela gramática.

2. Objetivo

O objetivo deste projeto é a criação de um analisador sintático para a linguagem MicroPascal, com a ajuda de um lexer (analisador léxico), que será responsável por transformar o código-fonte em tokens, e um parser, que irá verificar a validade desses tokens com base nas regras de produção.

3. Estruturas de Dados

3.1. Token

A struct Token é utilizada para representar os tokens extraídos do código-fonte. Cada token contém as seguintes informações:

- nome: Identifica o tipo do token (por exemplo, program, ID, NUM_INT, etc.).
- lexema: O valor real do token encontrado (por exemplo, program, x, 10, etc.).
- linha: A linha do código onde o token foi encontrado.
- coluna: A coluna do código onde o token foi encontrado.

Exemplo de uso:

```
Token token;
token.nome = "ID";
token.lexema = "x";
token.linha = 1;
token.coluna = 9;
```

3.2. Symbol

A struct Symbol é utilizada para representar os símbolos na tabela de símbolos. Cada símbolo tem:

- lexema: O nome do identificador.
- tipo: O tipo do símbolo, como RESERVED (palavra reservada) ou ID (identificador).

Exemplo de uso:

```
Symbol tabelaSimbolos[100];
```

4. Funções do Lexer

4.1. ehPalavraReservada

Verifica se o lexema fornecido é uma palavra reservada da linguagem. Retorna 1 se for, e 0 caso contrário.

4.2. ehSimbolo

Verifica se o lexema fornecido é um símbolo (como ;, ,, (,)).

4.3. inicializarTabelaSimbolos

Inicializa a tabela de símbolos, inserindo as palavras reservadas como símbolos do tipo RESERVED.

4.4. getToken

A função principal do lexer. Ela lê o código-fonte caractere por caractere e gera tokens conforme a gramática definida. Os tokens gerados incluem palavras reservadas, identificadores, números, operadores e símbolos.

5. Funções do Analisador Sintático

5.1. CasaToken

Verifica se o token atual é o esperado. Se for, avança para o próximo token. Caso contrário, exibe um erro.

5.2. Programa

Aplica a regra de produção programa -> 'program' ID ';' bloco '.' e chama as funções necessárias para validar cada parte dessa regra.

5.3. Bloco

Aplica a regra bloco -> declvar comando_composto e chama DeclVar e ComandoComposto.

5.4. DeclVar

Verifica se há declarações de variáveis e aplica a regra declvar -> 'var' lista_decl_var. Caso contrário, aplica a produção declvar -> ε.

5.5. ListaDeclVar

Aplica a regra lista_decl_var -> ID ':' tipo ';' lista_decl_var | ε para processar as declarações de variáveis.

5.6. Tipo

Verifica o tipo da variável, aplicando as produções tipo -> 'integer' ou tipo -> 'real'.

5.7. ComandoComposto

Aplica a regra comando_composto -> 'begin' lista_comandos 'end'.

5.8. ListaComandos

Aplica a regra lista_comandos -> comando ';' lista_comandos | ε, processando cada comando do programa.

5.9. Comando

Aplica a regra comando -> ID ':=' expressao ou comando -> comando_composto.

5.10. Expressao

Aplica a regra expressao -> termo e processa operadores adicionais (+, -, etc.).

5.11. Termo

Aplica a regra termo -> fator e lida com multiplicações e divisões.

5.12. Fator

Aplica a regra fator -> ID | NUM_INT | NUM_REAL | '(' expressao ')'.

6. Exemplos de Programas de Teste

6.1. Programas Corretos

```
program teste;
var
  x : integer;
begin
  x := 10;
end.
```

```
program teste;
var
  x : integer;
  y : real;
begin
  x := 5;
  y := 3.14;
end.
```

```
program teste;
var
```

```
  a : integer;
begin
  a := 10;
  begin
    a := a + 5;
  end;
end.
```

6.2. Programas Errados

1. Falta de Ponto e Vírgula:

```
program teste;
var
  x : integer
begin
  x := 10;
end.
```

2. Uso de Palavra Reservada Errada:

```
program teste;
var
  x : inteiro; // "inteiro" não é uma palavra reservada
begin
  x := 10;
end.
```

3. Falta de Palavra Reservada:

```
program teste;
  x := 10; // falta a palavra "var"
begin
  x := 10;
end.
```