Sumário

1.Introdução

Este relatório apresenta o desenvolvimento do **analisador sintático** da linguagem MicroPascal, correspondente à **segunda etapa** do trabalho prático. Após a implementação do analisador léxico (etapa anterior), esta fase teve como objetivo construir um **parser descendente recursivo**, capaz de validar a estrutura sintática completa de um programa na linguagem, conforme a gramática fornecida pelo professor.

A análise sintática consiste em verificar se a sequência de tokens produzida pelo analisador léxico obedece às regras formais da linguagem. Em caso de inconsistência, o compilador deve produzir mensagens de erro contendo número da linha e o lexema encontrado.

GitHub: https://github.com/EduardaCodes/Trabalho-TP02-COMP10.git

2. Implementação

A implementação utilizou a arquitetura clássica de compiladores: um analisador léxico responsável por gerar tokens e um analisador sintático que consome esses tokens conforme a gramática. As funções foram implementadas em C, sem uso de geradores automáticos.

2.1 Estruturas (structs)

• As principais estruturas utilizadas foram:

struct Token
Representa uma unidade léxica e contém:

- TipoToken tipo — categoria do token
- char lexema[] — texto original lido
- int linha — linha de origem
- int coluna — posição na linha

Enum TipoToken
Enumeração contendo todos os tokens definidos pela gramática MicroPascal, como:

- palavras-chave (TOKEN_PROGRAM, TOKEN_VAR, TOKEN_BEGIN ...)
- operadores (TOKEN_MAIS, TOKEN_ATRIBUICAO, TOKEN_MENOR_IGUAL ...)
- símbolos (TOKEN_PONTO_VIRGULA, TOKEN_DOIS_PONTOS ...)
- literais (TOKEN_NUMERO, TOKEN_TRUE, TOKEN_FALSE)

Essas estruturas estão declaradas no arquivo **tokens.h.**

2.2 Funções

• A implementação do analisador sintático foi feita no arquivo **sintatico.c,** contendo:

Funções gerais

- analisarSintatico(FILE*) — inicializa o léxico, lê o primeiro token e inicia a análise pelo símbolo inicial (programa).
- casaToken(TipoToken) — compara o token atual com o esperado; caso contrário, gera erro sintático.
- erroSintatico() — imprime mensagem detalhada de erro.

Não-terminais implementados (um por produção da gramática):

- programa()
- bloco()
- parteDeclaracoesVariaveis()
- declaracaoVariaveis()
- listaIdentificadores()
- tipo()

Comandos

- comando()
- comandoComposto()
- comandoCondicional() (if/then/else)
- comandoRepetitivo() (while/do)
- atribuicao()
- comandoEntrada() (read)
- comandoSaida() (write)

Expressões

- expressao()
- expressaoSimples()
- termo()
- fator()

A estrutura segue rigorosamente o modelo de analisador **top-down recursivo,** onde cada função corresponde a um não-terminal da gramática.

2.3 Autômato Finito Determinístico (AFD)

Diferente do trabalho anterior, o analisador sintático **não utiliza Autômato Finito Determinístico**, pois o AFD pertence exclusivamente ao analisador léxico. Nesta fase, todo o reconhecimento estrutural é baseado em **chamadas recursivas** e **verificação de tokens via casaToken()**.

3. Testes

Foram criados seis programas de teste: três válidos e três contendo erros sintáticos propositalmente.

3.1 Programas Corretos

Teste 1 - Atribuição Simples

```
program Teste1;

var x : integer;

begin

   x := 10;

end.
```

```
$ ./analisador testeCerto1.c
Analisando arquivo: testeCerto1.c

=== INICIANDO ANALISE SINTATICA ===

Token aceito: program
Token aceito: p1
Token aceito: ;
Token aceito: var
Token aceito: x
Token aceito: :
Token aceito: integer
Token aceito: ;
Token aceito: begin
Token aceito: x
Token aceito: :=
Token aceito: 10
Token aceito: ;
Token aceito: end
Token aceito: .

=== ANALISE SINTATICA CONCLUIDA COM SUCESSO ===
```

Teste 2 - IF/ELSE

```
program Teste2;

var x, y : integer;

begin

   x := 5;
```

```
    y := 10;


    if x < y then
        x := x + 1
    else
        y := y - 1;
end.
```

```
$ ./analisador testeCerto2.c
Analisando arquivo: testeCerto2.c

=== INICIANDO ANALISE SINTATICA ===

Token aceito: program
Token aceito: teste2
Token aceito: ;
Token aceito: var
Token aceito: x
Token aceito: ,
Token aceito: y
Token aceito: :
Token aceito: integer
Token aceito: ;
Token aceito: begin
Token aceito: x
Token aceito: :=
Token aceito: 5
Token aceito: ;
Token aceito: y
Token aceito: :=
Token aceito: 10
Token aceito: ;
Token aceito: if
Token aceito: x
Token aceito: <
Token aceito: y
Token aceito: then
Token aceito: x
Token aceito: :=
Token aceito: x
Token aceito: +
Token aceito: 1
Token aceito: else
Token aceito: y
Token aceito: :=
Token aceito: y
Token aceito: -
Token aceito: 1
Token aceito: ;
Token aceito: end
Token aceito: .

=== ANALISE SINTATICA CONCLUIDA COM SUCESSO ===
```

Teste 3 - While

```
program Loop;

var i : integer;

begin

    i := 0;
```

```
while i < 5 do
    i := i + 1;

end.
```

```
$ ./analisador testeCerto3.c
Analisando arquivo: testeCerto3.c

=== INICIANDO ANALISE SINTATICA ===

Token aceito: program
Token aceito: loop
Token aceito: ;
Token aceito: var
Token aceito: i
Token aceito: :
Token aceito: integer
Token aceito: ;
Token aceito: begin
Token aceito: i
Token aceito: :=
Token aceito: 0
Token aceito: ;
Token aceito: while
Token aceito: i
Token aceito: <
Token aceito: 5
Token aceito: do
Token aceito: i
Token aceito: :=
Token aceito: i
Token aceito: +
Token aceito: 1
Token aceito: ;
Token aceito: end
Token aceito: .

=== ANALISE SINTATICA CONCLUIDA COM SUCESSO ===
```

3.2 Programas com Erros Sintáticos

Erro 1 - Falta de ponto e vírgula nas declarações

```
program Erro1;

var x : integer

begin
    x := 10;

end.
```

```
=== INICIANDO ANALISE SINTATICA ===

Token aceito: program
Token aceito: erro1
Token aceito: ;
Token aceito: var
Token aceito: x
Token aceito: :
Token aceito: integer
ERRO SINTATICO [Linha 2, Coluna 16]: Token esperado diferente do encontrado
Token encontrado: begin
Token aceito: x
ERRO SINTATICO [Linha 4, Coluna 6]: Token esperado diferente do encontrado
Token encontrado: :=
ERRO SINTATICO [Linha 4, Coluna 9]: Tipo esperado (integer ou boolean)
Token encontrado: 10
ERRO SINTATICO [Linha 4, Coluna 9]: Token esperado diferente do encontrado
Token encontrado: 10
ERRO SINTATICO [Linha 4, Coluna 12]: Token esperado diferente do encontrado
Token encontrado: ;
Token aceito: end
Token aceito: .

=== ANALISE SINTATICA CONCLUIDA COM 5 ERRO(S) ===
```

Erro 2 - IF sem THEN

```
program Erro2;

var x : integer;

begin

   if x = 10

      x := 5;

end.
```

Erro 3 - Finalização incorreta (faltando end.)

```
program Erro3;

var x : integer;

begin

   x := 2;

   begin

      x := x + 1;

end.
```

4. Conclusão

A implementação do analisador sintático permitiu compreender a etapa de validação estrutural do processo de compilação, aplicando a técnica de **análise descendente recursiva**. O trabalho reforçou conceitos de gramática livre de contexto, tokens, chamadas recursivas e detecção de erros sintáticos.

Como melhorias futuras, poderíamos implementar:

- recuperação síncrona de erros via conjuntos FIRST/FOLLOW;
- produção opcional de árvore sintática abstrata (AST);
- integração com a etapa de análise semântica.

Referências

MICROPASCAL – Definição da linguagem. Material de apoio fornecido em sala.
Compiladores – Princípios, técnicas e ferramentas. Aho, Lam, Sethi, Ullman.

Anexos

- main.c
- lexico.c
- sintatico.c
- tokens.h
- Programas de teste (corretos e com erro)