UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Introdução à Compilação (11927/31)

Profa.ª Dra.ª Valéria Delisandra Feltrim

OLGA MARIA DOS SANTOS RA 130002

TRABALHO PRÁTICO Parte 2: Geração de código MEPA para a linguagem Tascal

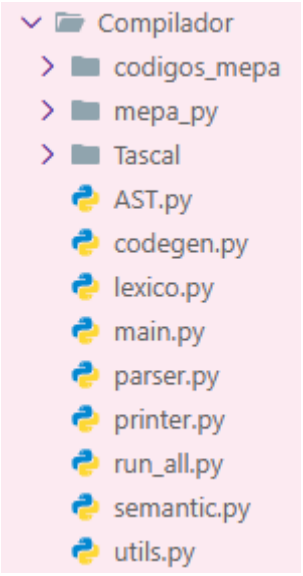
MARINGÁ

2025

Ferramentas utilizadas: PLY e MEPA.

Organização básica do código:

A organização possuí a seguinte estrutura:



Descrição do que cada arquivo .py é responsável por fazer

1. **Lexico:** Responsável pela análise léxica utilizando PLY. Define palavras reservadas, tokens, literais, expressões regulares para cada token da linguagem e a função de tratamento de erro léxico (t_error).
2. **Parser:** Implementa o analisador sintático com PLY/YACC. Define a gramática da linguagem Tascal, regras de precedência, construção da AST a partir das produções e tratamento de erros sintáticos.
3. **AST:** Contém as classes dos nós da Árvore Sintática Abstrata. Representa a estrutura lógica do programa, como comandos, expressões, operações, variáveis e blocos.
4. **Utils:** contém classes de utilidade “geral”, como classes de tipos, de símbolos para uso na tabela de símbolos e principalmente a visitor que é utilizada em todos os outros visitor para impressão da ast, para análise semântica e geração de código.
5. **Printer:** Percorre a AST utilizando o padrão Visitor e imprime a árvore de forma estruturada. Usado para depuração.
6. **Semantic:** Realiza a análise semântica do programa. Valida declarações, tipos, escopo, compatibilidade de operações, uso correto de variáveis e anota cada identificador com seu símbolo (incluindo deslocamento para geração de código). Reporta erros semânticos detalhados.

7. **Codegen:** Percorre a AST e gera o código MEPA correspondente. Traduz expressões, comandos, desvios condicionais, loops, leitura, escrita e atribuições para instruções da máquina virtual MEPA. Também gerencia rótulos, alocação e desalocação de memória.
8. **Main:** Ponto de entrada do compilador. Lê o arquivo de entrada, aplica os analisadores conforme a flag fornecida, imprime mensagens de sucesso/erro e, no modo -g, salva o código MEPA gerado em um arquivo .mep.

Para executar o comando é:

```python
python main.py <flag> Tascal/<arquivo>.tascal
```

Possibilidades para a flag:

```
-l : Executa apenas a análise léxica
-p : Executa as análises léxica e sintática
-pp : Executa as análises léxica e sintática e imprime e AST.
-s : Executa as análises léxica, sintática e semântica
-g : Executa o pipeline completo e gera o código e salva em um arquivo.
```

9. Run_all: Executa automaticamente o compilador (`main.py`) usando a flag definida no cabeçalho para todos os arquivos `.tascal` da pasta “Tascal”.

```
FLAG = "-g"        # -l, -p, -pp, -s, -g
```

```python
python run_all.py
```

- A pasta “Tascal” têm os programas de teste .tascal
- mepa_py é diretório da máquina virtual denominada MEPA (Kowaltowski, 1983).
- A pasta “codigos_mepa” é o diretório onde é salvo os arquivos de instruções .mep gerados pelo GeradorDeCodigo dentro de codegen.py, a partir da execução da main com a flag “-g”.

Etapas implementadas:

O compilador para a linguagem Tascal foi desenvolvido seguindo todas as fases clássicas de um compilador, desde a leitura do código-fonte até a geração do código intermediário para a máquina MEPA. As etapas implementadas foram:

1. **Análise Léxica:** Foi implementado um analisador léxico completo utilizando o PLY (Flex) com detecção de erros léxicos com indicação de linha;
2. **Análise Sintática:** A análise sintática foi implementada com PLY (yacc) com construção da Árvore Sintática Abstrata (AST) em cada produção; Todos os programas válidos foram corretamente reconhecidos, e os inválidos foram rejeitados;
3. **Construção da AST:** Foi seguido a indicação do trabalho e foi implementado uma AST explícita para modularizar melhor o compilador.;
4. **Análise Semântica:** A análise semântica percorre a AST e aplica todas as regras especificadas no trabalho. Erros semânticos são reportados com indicação de tipo e posição;
5. **Geração de Código MEPA:** Foi implementado um gerador de código intermediário capaz de traduzir todo programa Tascal para instruções MEPA;
6. **Execução Automática e Organização:** Foram implementadas funcionalidades adicionais que facilitam o uso e validação do compilador:
    - execução direta via linha de comando com flags (-l, -p, -pp, -s, -g);
    - geração automática de arquivos .mep na pasta codigos_mepa;
    - script run_all.py para processar automaticamente todos os .tascal;

Foi executado todos os programas .tascal, e todos com erro foram detectados e com a origem do erro correta (léxico, sintático ou semântico). Os 10 programas .tascal corretos foram gerados seus códigos intermediários para a MEPA e executados eles de acordo com as instruções em “Casos_Teste_Tascal” e todos obtiveram a saída esperada.

Comando para execução do código intermediário:

```python
python mepa_py/mepa_pt.py --progfile codigos_mepa/<arquivo_entrada>
```

Descrição e justificativas de possíveis etapas não cumpridas:

Todas as etapas foram cumpridas.