Universidade Estadual de Maringá

Centro de Tecnologia

Departamento de Informática

Informática

Introdução à Compilação

Trabalho 01

Professora: Valéria Feltrim

Alunos	RA
Leonardo Almenara	128432
Gustavo Michelim	128968

1- Ferramenta usada:

O compilador da linguagem Tascal foi desenvolvido em Python utilizando a biblioteca PLY (Python Lex-Yacc), que implementa em Python as funcionalidades das ferramentas clássicas Lex e Yacc, amplamente utilizadas na construção de compiladores. O PLY permite definir analisadores léxicos e sintáticos diretamente no código, por meio de expressões regulares e regras gramaticais escritas em docstrings, facilitando o desenvolvimento e a compreensão do processo de análise.

O módulo LEX realiza a análise léxica, identificando os tokens do código-fonte — como identificadores, operadores, números e palavras reservadas — por meio de expressões regulares. Já o módulo YACC executa a análise sintática, verificando se a sequência de tokens segue as regras da gramática da linguagem. Cada produção é implementada como uma função Python, o que permite associar ações semânticas diretamente às regras, como a instalação de variáveis na tabela de símbolos, a verificação de tipos e a detecção de usos incorretos de identificadores.

O PLY utiliza o método de análise LALR(1), o mesmo adotado por ferramentas como o Yacc e o Bison, garantindo eficiência e correção na interpretação da gramática. Ele gera automaticamente as tabelas de parsing e reconhece conflitos de precedência, permitindo o controle explícito de operadores e associatividade. Além disso, sua implementação puramente em Python torna possível integrar facilmente estruturas de dados e mensagens personalizadas ao processo de análise.

2- Organização básica do código:

A implementação do compilador Tascal foi organizada em módulos que representam as principais etapas do processo de compilação. O arquivo **lexer.py** contém o analisador léxico, desenvolvido com o módulo PLY.Nele são definidos os tokens da linguagem, utilizando expressões regulares para reconhecer identificadores, números, operadores, palavras reservadas e símbolos especiais. Esse módulo também realiza o tratamento de erros léxicos, exibindo mensagens informativas com a linha em que o símbolo inválido foi encontrado.

O arquivo **parser.py** implementa o analisador sintático e semântico por meio do módulo YACC. Ele define as regras gramaticais da linguagem Tascal e as precedências entre operadores, garantindo a interpretação correta das expressões. Nesse módulo também são realizadas as verificações semânticas, como o controle da tabela de símbolos, a checagem de variáveis não declaradas e a compatibilidade de tipos em atribuições e expressões. Ao final da análise, são exibidas mensagens indicando os erros detectados ou o sucesso do reconhecimento do programa.

Os arquivos **test_lexer.py** e **test_parser.py** são responsáveis pelos testes e execução das etapas de análise. O primeiro realiza a leitura de um código Tascal e exibe os tokens identificados, enquanto o segundo executa a análise sintática e semântica completa, reportando eventuais erros encontrados. Essa organização modular garante clareza,

facilidade de manutenção e separação adequada das fases léxica, sintática e semântica do compilador.

Para testar o trabalho é necessário rodar a partir da pasta raíz:

py -m "patch+arquivoTeste" "patch+nomeEntrada"

Exemplo:

py -m tascal_compiler.Tests.Parser.test_parser/ProgramasTascalTeste/P1.tascal

3- Ações Semânticas Implementadas

As ações semânticas do compilador Tascal, integradas ao módulo parser.py, realizam a verificação lógica do programa durante a análise sintática. Elas garantem a correção no uso de identificadores, tipos e expressões, utilizando uma tabela de símbolos (tabela_variaveis) para registrar as variáveis declaradas e seus tipos. Erros são armazenados em uma lista (erros_semanticos) e reportados ao final da execução. As ações implementadas foram:

- Inicialização semântica: função semantico_reset, que reinicia a tabela e a lista de erros a cada nova análise.
- Instalação e verificação de variáveis: instala_variavel adiciona identificadores à tabela, detectando duplicações. Busca_variavel garante que variáveis usadas estejam declaradas.
- Instalação do programa: instala_programa identifica o nome do programa principal, registrando seu início.
- Declaração de tipos: cada variável declarada herda o tipo definido (integer ou boolean), armazenado para checagens futuras.
- Atribuições: verifica compatibilidade entre o tipo da variável e o resultado da expressão.
- Comandos condicionais (if/else) e laços (while): exigem expressões booleanas como condição.
- Leitura e escrita: read() confirma se as variáveis passadas estão declaradas; write() valida se os tipos das expressões são inteiros ou booleanos.
- Expressões aritméticas: operações com +, -, * e div exigem operandos inteiros, e o resultado é do tipo integer.
- Expressões relacionais: operadores (=, <>, <, <=, >, >=) são verificados quanto à compatibilidade de tipos e retornam valor booleano.
- Expressões lógicas: operadores and, or e not requerem operandos booleanos, com verificação individual de cada caso.
- Operadores unários: o sinal - exige operandos inteiros, e not operandos booleanos.
- Expressões entre parênteses: mantêm o tipo da subexpressão interna, garantindo coerência semântica.
- Relato de erros: ao final da análise, todos os erros são listados com a linha correspondente, diferenciando falhas léxicas, sintáticas e semânticas.

Essas verificações asseguram que o compilador não apenas reconheça a estrutura sintática do programa, mas também valide seu significado semântico. Assim, o Tascal identifica erros de tipo, variáveis não declaradas e usos incorretos de operadores, fornecendo mensagens claras e precisas que auxiliam na correção do código-fonte.

4 - Descrição e justificativas de possíveis etapas não cumpridas

Este trabalho abrange apenas os analisadores Léxico, Sintático e Semântico. Nenhuma outra etapa do processo de desenvolvimento de um compilador foi realizada.

5 - Referências

- APPEL, A. W.; GINSBURG, M. Modern Compiler Implementation in C. Cambridge University Press, 1998.
- KOWALTOWSKI, T. Implementação de Linguagens de Programação. Guanabara Dois, 1983.
- PLY (Python Lex-Yacc) Documentation
- DUDUSCRIPT. pl0-ply: pl0 compiler written in python. https://github.com/duduscript/pl0-ply]
- STRIDERDU. *Plycc: A compiler for C language using PLY [https://github.com/striderdu/Plycc]
- SMOHAMMADFY. Compiler_PLY: Lexer and parser for compiler base like C/C++ with PLY python[https://github.com/smohammadfy/Compiler_PLY]