Trabalho de Compiladores Grau B

Alunos:

Giovani Fetzner

Renan Milech Pereira

Vicenzo Valmórbida

MiniCompiler - Python-Like Analyzer

- Projeto desenvolvido em Python
- Uso da biblioteca **PLY -> Substituindo a biblioteca SLY**
- **Pipeline:** Léxico → Sintaxe → AST → Semântica → Saídas (DOT, PNG, JSON)

Análise Léxica -> Objetivos

- Transformar o código-fonte em tokens estruturados
- Remover comentários, espaços irrelevantes e reconhecer padrões lexicais
- Lidar com indentação significativa (como Python), gerando INDENT e DEDENT
- Calcula indentação examinando espaços na linha seguinte
- Armazena indentação com **indent_stack**

Análise Léxica -> Tabela de Tokens

```
(base) C:\dev\MiniCompiler>python main.py --tokens testes/input2.txt
Tokens:
NAME    'a'     (lineno=1, pos=0)
=       '='     (lineno=1, pos=2)
NUMBER  10      (lineno=1, pos=4)
NEWLINE '\n'    (lineno=1, pos=6)
NAME    'b'     (lineno=2, pos=7)
=       '='     (lineno=2, pos=9)
NUMBER  3       (lineno=2, pos=11)
NEWLINE '\n'    (lineno=2, pos=12)
NAME    'c'     (lineno=3, pos=13)
=       '='     (lineno=3, pos=15)
NAME    'a'     (lineno=3, pos=17)
-       '-'     (lineno=3, pos=19)
NAME    'b'     (lineno=3, pos=21)
*       '*'     (lineno=3, pos=23)
NUMBER  2       (lineno=3, pos=25)
NEWLINE '\n'    (lineno=3, pos=26)
PRINT   'print' (lineno=4, pos=27)
(       '('     (lineno=4, pos=32)
NAME    'c'     (lineno=4, pos=33)
)       ')'     (lineno=4, pos=34)
NEWLINE '\n'    (lineno=4, pos=35)
```

Análise Sintática

- Converter a sequência de tokens em uma Árvore Sintática (AST)
- Validar a estrutura do programa segundo a gramática definida
- Detecção de erros: token inesperado, fim de arquivo prematuro, falta de : ou quebra de indentação
- Árvore montada utilizando método **LALR(1),** porem no código utilizado não teve lookahead

AST (abstract syntax tree) -> Nós

program – nó raiz do arquivo

assign – atribuições (x = ...)

var – identificadores

number – números literais

binop – operações (+, -, *, /)

print – chamada print

if – estrutura condicional

If_else – condicional com else

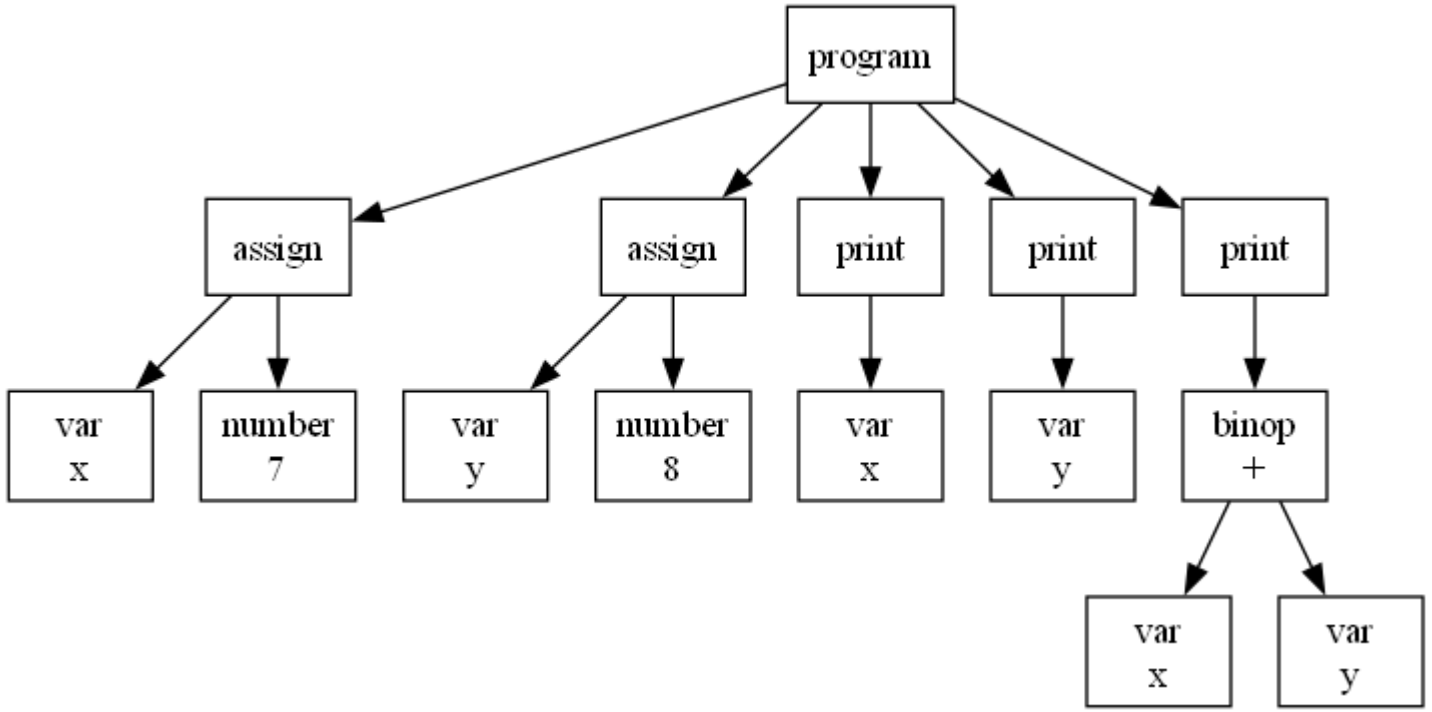
while – laço

block – bloco identado

AST (abstract syntax tree) -> Exemplo

```
x = 7
y = 8
print(x)
print(y)
print(x + y)
```

AST (abstract syntax tree) -> Exemplo



Análise Semântica -> Objetivos

- Garantir a **consistência** do programa:
- Tipos
- Variáveis declaradas corretamente
- **Regras:** Variáveis precisam ser declaradas antes do uso.
- Operações aritméticas só com tipos compatíveis .Tabela de símbolos atualiza tipos e categorias
- Resultado é o arquivo symbol_table.json

Análise Semântica -> Erros

```
Erro semantico: soma requer dois numeros ou duas strings na operacao '+' (tipos: string e number)
```

```
Erro semantico: operador logico requer booleanos na operacao 'and' (tipos: boolean e number)
```

```
Erro semantico: operador 'not' requer boolean, recebeu number
```

```
Erro semantico: operador '>' aceita apenas comparacao numerica na operacao '>' (tipos: number e string)
```

UNISINOS
DESAFIE O AMANHÃ.

Percurso da AST ou Execução

Fluxo no modo **--run**

Lexer -> Parser AST -> Semântica -> Exporta: Ast.dot, ast.png, symbol_table.json

Fluxo no modo **--token**

Lexer

Fluxo no modo **--ast**

Lexer -> Parser AST Exporta: Ast.dot, ast.png

Arquitetura do Projeto

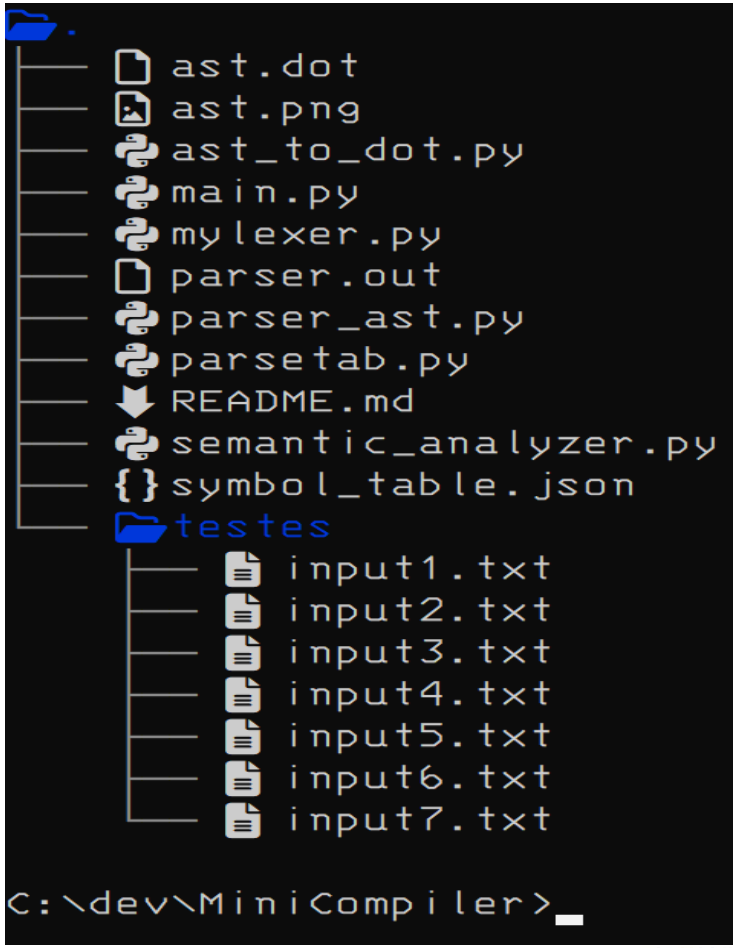
Main.py: orquestra o pipeline completo

Mylexer.py: implementa o analisador léxico com PLY, gera tokens e controla indentação, Classe principal: PythonLikeLexer

parser_ast.py: define a gramática e monta a AST. Classes: PythonLikeParser, ASTNode

semantic_analyzer.py: Realiza a análise semântica, infere tipos e constrói a tabela de símbolos

ast_to_dot.py: Converte a AST para arquivo DOT utilizado para gerar o PNG



UNISINOS
DESAFIE O AMANHÃ.

Demonstração

Limitações

- **Gramática parcial**: muitas keywords estão no lexer mas não têm regra no parser; ainda há 2 conflitos shift/reduce e vários tokens não usados (avisos do PLY)

- **Semântica simples**: tipos básicos (number/string/boolean), sem escopos, sem funções, sem tipos compostos; não há inferência avançada nem verificação de redeclaração/uso fora de escopo

- **Ferramentas externas**: para visualizar a AST em PNG precisa do GraphViz instalado; sem isso só sai o DOT

Próximos passos e lições aprendidas

- Limpar gramática/tokens: remover keywords não usadas e resolver conflitos shift/reduce
- Expandir semântica: escopos, redeclaração, tipos compostos e regras para funções/chamadas
- **Lições aprendidas**: manter a gramática enxuta evita conflitos, um compilador real é algo extremamente complexo e foi uma das maiores revoluções da **computação**