

Tarefa 06: tradução de atribuições e expressões

GRUPO 7:

KAUA MACHADO DA SILVA

YASMINE MARTINS DA COSTA E SILVA

FELIPE LAGES DE LIMA

Construir um analisador sintático (parser) que:

- Receba como entrada um código-fonte simples da linguagem descrita.
- Reconheça declarações de variáveis, expressões aritméticas (binárias e unárias) e atribuições.
- Gere um **código intermediário de três endereços** para as expressões e atribuições.

Linguagem fonte reconhecida

1. Declaração de variáveis inteiras

Exemplo:

```
number a;  
number b;  
number c;
```

Aqui, "number" é a palavra-chave para declarar variáveis inteiras.

2. Expressões aritméticas

- Binárias: soma (+), subtração (-), multiplicação (*), divisão (/)
- Unárias: plus (+) e minus (-) unários

Exemplo de expressão que pode aparecer na atribuição:

```
b + -c #operação unária
```

3. Atribuições

Exemplo:

```
a = b + -c;
```

Linguagem destino (código intermediário)

Gerar **Código de três endereços** é uma forma de código intermediário onde cada instrução tem no máximo três operandos, algo assim:

```
t0 = minua c;  
t1 = b + t0;  
a = t1;
```

Resultado:

1. Especificações Léxicas e Sintáticas (TresEnderecos.g4)

```
grammar TresEnderecos;  
  
// Regras parser (sintaxe)  
  
program : instrucoes* EOF ;  
  
NUMBER : 'number' ;  
SEMI : ';' ;  
EQUAL : '=' ;  
MUL : '*' ;  
DIV : '/' ;  
ADD : '+' ;  
SUB : '-' ;  
LPAREN : '(' ;  
RPAREN : ')' ;  
  
instrucoes : declaracaoVariavel  
            | atribuicao;
```

```

declaracaoVariavel : 'number' ID ';' ;

atribuicao : ID '=' expr ';' ;

expr: expr op=('*'|'/') expr    # MulDiv
    | expr op=('+'|'-') expr    # AddSub
    | op=('+'|'-') expr         # UnaryOp
    | ID                        # Id
    | INT                       # Int
    | '(' expr ')'              # Parens
    ;

// Regras lexer (tokens)

ID    : [a-zA-Z_][a-zA-Z_0-9]* ;
INT   : [0-9]+ ;

WS    : [ \t\r\n]+ → skip ;

```

2. **Gerar o parser com ANTLR:** Usar o ANTLR para transformar as especificações em código Java que reconhece o código-fonte.

```
java -jar antlr-4.13.2-complete.jar -Dlanguage=Python3 .\TresEnderecos.g
```

3. **Ler arquivo de código-fonte:** O parser deve receber o código-fonte em um arquivo, analisar ele e gerar o código intermediário na saída padrão (terminal).

- `TresEnderecosListener.py`

```

from antlr4 import *
if "." in __name__:
    from TresEnderecosParser import TresEnderecosParser
else:
    from TresEnderecosParser import TresEnderecosParser

class TresEnderecosListener(ParseTreeListener):

    def __init__(self):

```

```

self.temp_count = 0 #contador de temporários,
self.code = []      # Armazena as instruções geradas
self.values = {}    # Mapeia nós da AST para valores intermediários

# Função auxiliar para criar novos variáveis temporários
def new_temp(self):
    self.temp_count += 1
    return f"t{self.temp_count}"

# =====
# Métodos de Programa
# =====
# Inicializa estruturas de dados ao entrar no programa
def enterProgram(self, ctx: TresEnderecosParser.ProgramContext):
    self.code.clear()
    self.temp_count = -1
    self.values.clear()

# Imprime o código gerado ao sair do programa
def exitProgram(self, ctx: TresEnderecosParser.ProgramContext):
    print("\n".join(self.code))

# =====
# Métodos 'exit'
# =====
# Saindo da regra instrucoes // Declarações não geram código no destino
def exitInstrucoes(self, ctx: TresEnderecosParser.InstrucoesContext):
    pass

# Saindo da declaração de variável // Declarações não geram código no destino
def exitDeclaracaoVariavel(self, ctx: TresEnderecosParser.DeclaracaoVariavelContext):
    pass

# Atribuição: id = expr;
def exitAtribuicao(self, ctx: TresEnderecosParser.AtribuicaoContext):
    var_name = ctx.ID().getText()
    expr_node = ctx.expr()
    expr_val = self.values[expr_node]

```

```

self.code.append(f"{var_name} = {expr_val}")

# Métodos de Expressões
# Gera código para operações unárias: "t = -expr"
def exitUnaryOp(self, ctx: TresEnderecosParser.UnaryOpContext):
    # expr_node = ctx.expr()
    # expr_val = self.values[expr_node]
    # op = ctx.op.text
    # temp = self.new_temp()
    # self.code.append(f"{temp} = {op}{expr_val}")
    # self.values[ctx] = temp

    """Gera código para operações unárias (minus e plus)"""
    expr_val = self.values[ctx.expr()] # Obtém o valor da expressão
    op = ctx.op.text                  # Obtém o operador ('+' ou '-')

    if op == '+':
        temp = self.new_temp()
        self.code.append(f"{temp} = plus {expr_val}")
        self.values[ctx] = temp
    else:
        temp = self.new_temp()
        self.code.append(f"{temp} = minus {expr_val}")
        self.values[ctx] = temp

# Gera código para multiplicação/divisão: "t = left * right"
def exitMulDiv(self, ctx: TresEnderecosParser.MulDivContext):
    left_val = self.values[ctx.expr(0)]
    right_val = self.values[ctx.expr(1)]
    op = ctx.op.text
    temp = self.new_temp()
    self.code.append(f"{temp} = {left_val} {op} {right_val}")
    self.values[ctx] = temp

# Gera código para adição/subtração: "t = left + right"
def exitAddSub(self, ctx: TresEnderecosParser.AddSubContext):
    left_val = self.values[ctx.expr(0)]

```

```

right_val = self.values[ctx.expr(1)]
op = ctx.op.text
temp = self.new_temp()
self.code.append(f"{temp} = {left_val} {op} {right_val}")
self.values[ctx] = temp

# Propaga valor de expressões entre parênteses
def exitParens(self, ctx: TresEnderecosParser.ParensContext):
    self.values[ctx] = self.values[ctx.expr()]

# Registra identificadores: valor é o próprio nome
def exitId(self, ctx: TresEnderecosParser.IdContext):
    self.values[ctx] = ctx.ID().getText()

# Registra valores inteiros: valor é o próprio número
def exitInt(self, ctx: TresEnderecosParser.IntContext):
    self.values[ctx] = ctx.INT().getText()

# =====
# Métodos 'enter'
# =====
# Métodos 'enter' são chamados ao ENTRAR em cada regra.
# Como não precisamos fazer nada antes, deixamos 'pass'.

```

- `main.py`

```

from antlr4 import *
from TresEnderecosLexer import TresEnderecosLexer
from TresEnderecosParser import TresEnderecosParser
from TresEnderecosListener import TresEnderecosListener

def main(file_path):
    input_stream = FileStream(file_path)
    lexer = TresEnderecosLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = TresEnderecosParser(stream)
    tree = parser.program()

```

```
listener = TresEnderecosListener()
walker = ParseTreeWalker()
walker.walk(listener, tree)

if __name__ == "__main__":
    import sys
    main(sys.argv[1]) # python main.py .\teste1.txt → Terminal
    print("\n")
```

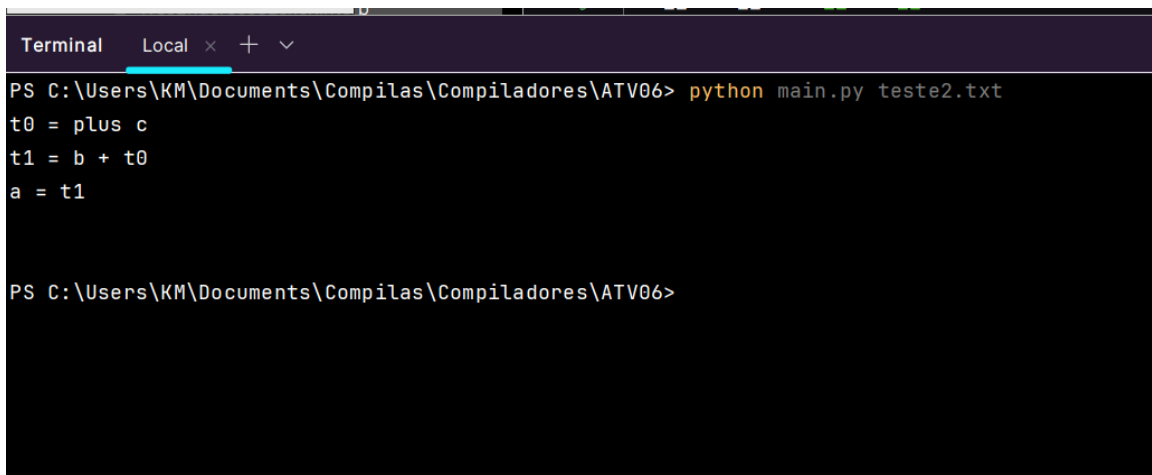
1. **Exemplo:** Dado o código:

```
python main.py teste1.txt #→ Terminal
```

- Entrada

```
number a;
number b;
number c;
a = b + +c;
```

- Saída:



```
Terminal Local x + v
PS C:\Users\KM\Documents\Compilas\Compiladores\ATV06> python main.py teste2.txt
t0 = plus c
t1 = b + t0
a = t1

PS C:\Users\KM\Documents\Compilas\Compiladores\ATV06>
```