



**UNIVERSIDADE ESTADUAL DE SANTA CRUZ DEPARTAMENTO
DE CIÊNCIAS EXATAS - DCET**

Adriel Fabrício da Silva Oliveira

**Relatório de implementação de um gerenciador de escopos em
Compiladores (DEC000058).**

**ILHÉUS - BAHIA
2024**

Adriel Fabrício da Silva Oliveira

**Relatório de implementação de um gerenciador de escopos em
Compiladores (DEC000058).**

Relatório de implementação de um gerenciador de escopos e verificação de tipos por meio de tabela de símbolos.

Orientador: Jacqueline Midlej Do Espirito Santo.

**ILHÉUS - BAHIA
2024**

1 Introdução

O documento descreve uma linguagem fictícia simplificada usada para atribuições de valores e criação de escopos. Esta linguagem tem comandos para iniciar e terminar blocos de código (BLOCO e FIM), declarações de variáveis (NUMERO e CADEIA), e comandos de impressão (PRINT). A linguagem é fortemente tipada e não permite o uso de variáveis não declaradas ou a conversão de tipos.

2 Definições das Expressões Regulares

2.1 Espaços em branco:

```
1 self.regex_whitespace = r"\s+"
2
```

Este padrão corresponde a um ou mais caracteres de espaço em branco.

2.2 Números:

```
1 self.regex_number = r"[+-]?\d+(\.\d+)?"
2
```

Este padrão corresponde a números inteiros ou reais, incluindo números positivos e negativos.

2.3

Atribuição de número a uma variável:

```
1 self.regex_variable_number = r"[a-zA-Z_][a-zA-Z0-9_]*\s*=\s*" + self.regex_number
2
```

Este padrão corresponde à atribuição de um número a uma variável.

2.4 Cadeias de caracteres:

```
1 self.regex_string = r'"([^"]*)"'
2
```

Este padrão corresponde a cadeias de caracteres delimitadas por aspas duplas.

2.5 Atribuição de cadeia a uma variável:

```
1 self.regex_variable_string = r"[a-zA-Z_][a-zA-Z0-9_]*\s*=\s*" + self.regex_string
2
```

Este padrão corresponde à atribuição de uma cadeia de caracteres a uma variável.

2.6 Atribuição entre variáveis:

```
1 self.regex_variable = r"[a-zA-Z_][a-zA-Z0-9_]*\s*=\s*[a-zA-Z_][a-zA-Z0-9_]*"
2
```

Este padrão corresponde à atribuição de uma variável a outra.

2.7 Comando de impressão:

```
1 self.regex_print = r"PRINT\s+[a-zA-Z_][a-zA-Z0-9_]*"  
2
```

Este padrão corresponde ao comando de impressão seguido de uma variável.

2.8 Declaração de número:

```
1 self.regex_declared_number = f"NUMERO{self.regex_whitespace}{self.regex_variable_number}"  
2
```

Este padrão corresponde à declaração e atribuição de um número a uma variável.

2.9 Declaração de número não atribuído:

```
1 self.regex_undeclared_number = r"NUMERO\s+[a-zA-Z_][a-zA-Z0-9_]*"  
2
```

Este padrão corresponde à declaração de um número sem atribuição inicial.

2.10 Declaração de cadeia:

```
1 self.regex_declared_string = f"CADEIA{self.regex_whitespace}{self.regex_variable_string}"  
2
```

Este padrão corresponde à declaração e atribuição de uma cadeia de caracteres a uma variável.

2.11 Declaração de cadeia não atribuída:

```
1 self.regex_undeclared_string = r"CADEIA\s+[a-zA-Z_][a-zA-Z0-9_]*"  
2
```

Este padrão corresponde à declaração de uma cadeia de caracteres sem atribuição inicial.

3 Gestão de Pilhas de Escopo

3.1 Processamento de Linhas

O método *process_line* é o núcleo da análise e manipulação de escopos. Ele realiza as seguintes operações com base na linha de código recebida:

1. Início de Bloco:
 - Detecta linhas que indicam o início de um novo bloco (BLOCO <identificador>).
 - Adiciona um novo escopo (lista vazia) à lista *scopes*.
 - Adiciona uma linha de início de bloco às *output_lines*.
2. Fim do Bloco:
 - Detecta linhas que indicam o fim de um bloco (FIM <identificador>).
 - Remove o escopo mais interno da lista *scopes*.
 - Adiciona uma linha de fim de bloco às *output_lines*.
3. Declaração e Atribuição de Variáveis:
 - Processa declarações e atribuições de variáveis numéricas e de cadeias de caracteres, tanto declaradas quanto não declaradas.
 - Separa múltiplas atribuições na mesma linha.
 - Verifica se a variável já existe no escopo atual antes de adicionar um novo token.
4. Atribuições Simples:
 - Processa atribuições simples de variáveis (<var1> = <var2>).
 - Verifica a existência das variáveis e a compatibilidade de tipos antes de realizar a atribuição.
5. Comando de Impressão:
 - Processa comandos de impressão (PRINT <variável>).
 - Verifica se a variável foi declarada e adiciona o valor da variável às *output_lines*.

3.2 Pilha de Escopos

A lista *scopes* é utilizada como uma pilha onde cada entrada representa um escopo. Novos escopos são empilhados no início de blocos e desempilhados no fim de blocos. Isto garante que variáveis e tokens sejam gerenciados de acordo com as regras de escopo, permitindo a correta visibilidade e manipulação de variáveis em diferentes níveis de aninhamento.

4 Implementação

4.1 Passo a Passo do Código

4.1.1 Importações e Definição da Classe

A classe *ScopeManager* é responsável por gerenciar escopos e tokens. Ela usa expressões regulares para identificar diferentes padrões na linguagem, como números, cadeias de caracteres, variáveis, e comandos de impressão. Além disso, mantém uma lista de escopos e identificadores de bloco.

4.1.2 Inicialização

O método `__init__` inicializa as expressões regulares para reconhecer padrões na linguagem e configura as listas para gerenciar escopos e identificadores de bloco.

4.1.3 Métodos Auxiliares

Os métodos auxiliares *get_token_by_identifier*, *variable_exists*, e *assign_value_to_token* ajudam a gerenciar a tabela de símbolos, buscando tokens por identificador, verificando a existência de variáveis e atribuindo valores a tokens.

4.1.4 Processamento de Linhas

O método *process_line* processa cada linha do código de entrada. Ele reconhece e trata diferentes tipos de declarações e atribuições, comandos de início e fim de blocos, e comandos de impressão. Ele utiliza expressões regulares para identificar padrões específicos e manipula a tabela de símbolos conforme necessário.

4.1.5 Processamento de Arquivo

O método *process_scope* lê um arquivo linha por linha e chama *process_line* para cada linha. O código é executado a partir do método `__main__`, que cria uma instância de *ScopeManager* e chama *process_scope* com um caminho de arquivo de entrada.

5 Pseudocódigo

Algorithm 1 Classe ScopeManager

```
1: Atributos:
2:   regex_whitespace, regex_number, regex_variable_number
3:   regex_string, regex_variable_string, regex_variable
4:   regex_print, regex_declared_number, regex_undeclared_number
5:   regex_declared_string, regex_undeclared_string
6:   scopes, output_lines, block_identifiers
7: procedure __INIT__
8:   Inicializar expressões regulares para corresponder a diferentes padrões
9:   Inicializar scopes, output_lines e block_identifiers
10: procedure GET_TOKEN_BY_IDENTIFIER(identifier)
11:   for cada escopo em ordem reversa do
12:     for cada token no escopo do
13:       if identificador do token corresponder then
14:         return token
15:   return None
16: procedure VARIABLE_EXISTS(identifier, scope)
17:   if algum token no escopo possui o identificador fornecido then
18:     return True
19:   else
20:     return False
21: procedure ASSIGN_VALUE_TO_TOKEN(identifier, value)
22:   for cada escopo em ordem reversa do
23:     for cada token no escopo do
24:       if identificador do token corresponder then
25:         Atualizar seu valor
26: procedure PROCESS_LINE(line, line_number)
27:   Limpar e formatar a linha
28:   if linha iniciar um bloco then
29:     Adicionar identificador do bloco a block_identifiers
30:     Adicionar um novo escopo a scopes
31:     Adicionar um marcador de início de bloco a output_lines
32:   else if linha finalizar um bloco then
33:     Remover o último escopo de scopes
34:     Adicionar um marcador de fim de bloco a output_lines
35:     Remover identificador do bloco de block_identifiers
36: procedure PROCESS_SCOPE(file_path)
37:   Abrir o arquivo e ler as linhas
38:   for cada linha no arquivo do
39:     Chamar PROCESS_LINE(linha)
40:   Garantir que o diretório de saída exista
41:   Escrever output_lines em um arquivo de saída
42: procedure MAIN_EXECUTION
43:   Analisar argumentos da linha de comando para obter o caminho do arquivo
44:   Criar uma instância de ScopeManager
45:   Chamar PROCESS_SCOPE(caminho do arquivo)
```

6 Testes - Entradas e Saídas

O programa recebe como entrada um arquivo, exemplo:

```
[language=bash]  
python3 scope_manager.py inputs/sample_01.txt
```

E a saída dele é criada na pasta *outputs/* com extensão *.out*. O nome do arquivo de saída é o mesmo que o de entrada. Neste exemplo a saída sera: *sample_01.out*.

6.1 Teste 01

6.1.1 Entrada

```
BLOCO _principal_  
NUMERO a = 10, b = 20  
CADEIA x  
  
PRINT b  
PRINT a  
x= "Ola mundo"  
x=a  
PRINT x  
BLOCO _n1_  
CADEIA a = "Compiladores"  
NUMERO c  
c=-0.45  
PRINT b  
PRINT c  
FIM _n1_  
  
BLOCO _n2_  
CADEIA b = "Compiladores"  
PRINT a  
PRINT b  
a=11  
CADEIA a= "Bloco2"  
PRINT a  
PRINT c  
BLOCO _n3_  
NUMERO a=-0.28, c  
PRINT a  
PRINT b  
PRINT c  
c=a  
PRINT c  
a=40  
PRINT a  
print c  
FIM _n3_  
FIM _n2_  
PRINT c  
PRINT a  
FIM _principal_
```

6.1.2 Saída

```
*INICIO _principal_*  
b = 20 em _principal_  
a = 10 em _principal_  
Linha 8: x : Atribuicao invalida  
x = "Ola mundo" em _principal_
```

```
*INICIO _n1_*  
b = 20 em _n1_  
c = -0.45 em _n1_
```

```
*FIM _n1_*
```

```
*INICIO _n2_*  
a = 10 em _n2_  
b = "Compiladores" em _n2_  
a = "Bloco2" em _n2_  
Linha 25: c nao declarado
```

```
*INICIO _n3_*  
a = -0.28 em _n3_  
b = "Compiladores" em _n3_  
c = 0 em _n3_  
c = -0.28 em _n3_  
a = 40 em _n3_
```

```
*FIM _n3_*
```

```
*FIM _n2_*  
Linha 38: c nao declarado  
a = 11 em _principal_
```

```
*FIM _principal_*
```

6.2 Teste 02

6.2.1 Entrada

```
BLOCO _principal_  
NUMERO a_1 = 12345, b, c=0.2345  
CADEIA x=""  
x=a
```

```
BLOCO _n1_  
a_1=45  
b=55  
c=25  
x=15
```

```
NUMERO nova=10  
FIM _n1_
```

```
PRINT a_1  
PRINT nova
```

```
BLOCO _n2_  
PRINT nova  
NUMERO x=42, c  
c=x  
x=90  
PRINT c  
PRINT x  
FIM _n2_
```

```
PRINT x
```

```
BLOCO _n3_  
PRINT nova  
CADEIA x="Nova cadeia"  
PRINT x  
BLOCO _n4_  
PRINT x  
PRINT c  
PRINT a  
NUMERO a=81  
FIM _n4_  
PRINT a  
FIM _n3_
```

```
PRINT nova  
b=-934.0
```

```

PRINT b
PRINT a
a=b
PRINT a
FIM _principal_

```

6.2.2 Saída

```

*INICIO _principal_*
Linha 4: a nao declarado

*INICIO _n1_*
Linha 10: x : Atribuicao invalida

*FIM _n1_*
a_1 = 45 em _principal_
Linha 16: nova nao declarado

*INICIO _n2_*
Linha 19: nova nao declarado
c = 42 em _n2_
x = 90 em _n2_

*FIM _n2_*
x = "" em _principal_

*INICIO _n3_*
Linha 30: nova nao declarado
x = "Nova cadeia" em _n3_

*INICIO _n4_*
x = "Nova cadeia" em _n4_
c = 25 em _n4_
Linha 36: a nao declarado

*FIM _n4_*
Linha 39: a nao declarado

*FIM _n3_*
Linha 42: nova nao declarado
b = -934.0 em _principal_
Linha 45: a nao declarado
a = -934.0 em _principal_

*FIM _principal_*

```

6.3 Teste 03

6.3.1 Entrada

```
BLOCO _principal_  
NUMERO a = 12345  
PRINT a
```

```
BLOCO _n1_  
a=45  
NUMERO a=25  
PRINT a
```

```
BLOCO _n2_  
CADEIA a="Compiladores"  
PRINT a  
FIM _n2_
```

```
PRINT a
```

```
BLOCO _n3_  
a="Compiladores"  
PRINT a  
BLOCO _n4_  
CADEIA a  
PRINT a  
a="Hello"  
PRINT a  
a=10  
PRINT a  
FIM _n4_  
FIM _n3_
```

```
FIM _n1_
```

```
PRINT a
```

```
FIM _principal_
```

6.3.2 Saída

```
*INICIO _principal_*  
a = 12345 em _principal_
```

```
*INICIO _n1_*  
a = 25 em _n1_
```

```
*INICIO _n2_*  
a = "Compiladores" em _n2_  
  
*FIM _n2_*  
a = 25 em _n1_  
  
*INICIO _n3_*  
Linha 18: a : Atribuicao invalida  
a = 25 em _n3_  
  
*INICIO _n4_*  
a = 0 em _n4_  
a = "Hello" em _n4_  
Linha 25: a : Atribuicao invalida  
a = "Hello" em _n4_  
  
*FIM _n4_*  
  
*FIM _n3_*  
  
*FIM _n1_*  
a = 45 em _principal_  
  
*FIM _principal_*
```

6.4 Teste 04

6.4.1 Entrada

```
BLOCO _principal_  
NUMERO a = 10, b = 20  
CADEIA x  
  
PRINT b  
PRINT a  
x= "Ola mundo"  
x=a  
PRINT x  
BLOCO _n1_  
CADEIA a = "Compiladores"  
NUMERO c  
c=-0.45  
PRINT b  
PRINT c  
FIM _n1_  
  
BLOCO _n2_  
CADEIA b = "Compiladores"  
PRINT a  
PRINT b  
a=11  
CADEIA a= "Bloco2"  
PRINT a  
PRINT c  
  
BLOCO _n3_  
NUMERO a=-0.28, c  
PRINT a  
PRINT b  
PRINT c  
c=a  
PRINT c  
a=40  
PRINT a  
print c  
FIM _n3_  
FIM _n2_  
PRINT c  
PRINT a  
FIM _principal_
```


6.4.2 Saída

```
*INICIO _principal_*  
b = 20 em _principal_  
a = 10 em _principal_  
Linha 8: x : Atribuicao invalida  
x = "Ola mundo" em _principal_
```

```
*INICIO _n1_*  
b = 20 em _n1_  
c = -0.45 em _n1_
```

```
*FIM _n1_*
```

```
*INICIO _n2_*  
a = 10 em _n2_  
b = "Compiladores" em _n2_  
a = "Bloco2" em _n2_  
Linha 25: c nao declarado
```

```
*INICIO _n3_*  
a = -0.28 em _n3_  
b = "Compiladores" em _n3_  
c = 0 em _n3_  
c = -0.28 em _n3_  
a = 40 em _n3_
```

```
*FIM _n3_*
```

```
*FIM _n2_*  
Linha 39: c nao declarado  
a = 11 em _principal_
```

```
*FIM _principal_*
```

7 Conclusão

A classe *ScopeManager* implementa a gestão de escopos, manipulando corretamente a visibilidade e a vida útil das variáveis através de uma pilha de escopos. Isso é fundamental em compiladores e interpretadores para garantir que variáveis e outras entidades sejam gerenciadas conforme as regras da linguagem de programação.