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:

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

```

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

2.2 Números:

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

```

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

2.3

Atribuição de número a uma variável:

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

```

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

2.4 Cadeias de caracteres:

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

```

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

2.5 Atribuição de cadeia a uma variável:

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

```

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

2.6 Atribuição entre variáveis:

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

```

Este padrão corresponde à atribuição de uma variável a outra.

2.7 Comando de impressão:

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

```

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

2.8 Declaração de número:

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

```

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

2.9 Declaração de número não atribuído:

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

```

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

2.10 Declaração de cadeia:

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

```

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

2.11 Declaração de cadeia não atribuída:

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

```

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 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*.

5.1 Teste 01

5.1.1 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_
```

5.1.2 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_*
```

5.2 Teste 02

5.2.1 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_
```

5.2.2 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_*
```

5.3 Teste 03

5.3.1 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_
```

5.3.2 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_*
```

5.4 Teste 04

5.4.1 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_
```

5.4.2 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_*
```

6 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.