um.jpg

Universidade do Minho

Licenciatura em Ciências da Computação

PLC - Trabalho Prático 2

Grupo nº14

Simão Pedro Batista Caridade Quintela
(A97444)

David José de Sousa Machado
(A91665)

Hugo Filipe de Sá Rocha
(A96463)

14 de Janeiro de 2023



Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Rangel Henriques o desenvolvimento de uma Linguagem de Programação Imperativa simples e de um compilador para reconhecer programas escritas nessa linguagem gerando o respetivo código Assembly da Máquina Virtual VM.

Começamos por tentar encontrar um nome original e atrativo para a nossa linguagem e acabou por nos surgir a ideia de colocar o nome "Python-Like-C"cuja sigla (PLC) coincide com a sigla da Unidade Curricular que integra este trabalho (Processamento de Linguagens e Compiladores).

Neste documento está apresentada a gramática da nossa linguagem, o código escrito no módulo Lexer e Yacc do Python e ainda, como foi pedido, alguns testes com código escrito na nossa linguagem e o respetivo código Assembly gerado.

Capítulo 2

Explicação da nossa linguagem

A nossa linguagem contempla os seguintes mecanismos:

2.1 Declaração de variáveis

```
1
2       int x = 10
3       int x
4       int x = 10
5       int x[n]
6       int x[n][m]
7
```

2.2 Operadores de comparação

```
1
2       x <= y
3       x >= y
4       x < y
5       x < y
6       x == y
7
```

2.3 Operações numéricas

```
1
2      x + y
3      x - y
4      x / y
5      x * y
6      x % y
7      x ++ #(incremento)
8      y -- #(decremento)
```

```

```

2.4 Operadores lógicos

```

 x and y
 x or y

```

2.5 Instruções condicionais

```

if (cond):
elif (cond):
else:

```

2.6 Ciclo do-while

```

do:
...
...
while(cond):

```

2.7 Input/Output

```

x = input()
x = input("Declare the variable with the value: ")
print("Hello world!")

```

2.8 Comentário

```

#isto e um comentario na nossa linguagem

```

Capítulo 3

Resolução

3.1 Gramática

— Colocar nossa gramática —

3.2 Módulo Lexer

Neste módulo identificamos os diferentes tokens (símbolos terminais da gramática) e a expressão regular que os caracteriza. Foi também neste módulo que implementamos a indentação da nossa linguagem.

3.2.1 Tokens

Os tokens que definimos foram: – colocar tokens –

3.2.2 Expressões regulares que caracterizam os tokens

As expressões regulares que caracterizam os nossos tokens são: – colocar as funcoes dos tokens –

3.2.3 Indentação

– colocar indentação

3.3 Módulo Yacc

É neste módulo que implementamos a nossa gramática, isto é, onde caracterizámos cada produção da gramática. Abaixo indicámos as diversas produções da nossa gramática organizadas por tópico.

3.3.1 Programa

–colocar produçoes Programa :

3.3.2 Corpo

–colocar produçoes Corpo :

3.3.3 Newline

–colocar produçoes Newline :

3.3.4 Declarações

–colocar produçoes Decl :

3.3.5 Procedimentos

–colocar produçoes Proc :

3.3.6 If-Else

–colocar produçoes If :

3.3.7 Indentação

–colocar produçoes Dedent :

3.3.8 Operadores de comparação e operadores lógicos

–colocar produçoes Cond :

3.3.9 Atribuições

–colocar produçoes Atrib :

3.3.10 Print

–colocar produçoes Print :

3.3.11 Operadores aritméticos

–colocar produçoes Expr :

3.3.12 Input

–colocar produçoes Input :

3.3.13 Strings

–colocar produçoes String :

3.3.14 Erros

```
1
2       def p_error(p):
3           print('Syntax error!\np -> ', p)
4           parser.sucesso = False
5
```

Definimos também a precedência dos operadores aritméticos para que o cálculo de uma expressão aritmética seja feito de acordo com a precedência habitual dos operadores:

```
1
2           precedence = (
3               ("left", "SUM", "SUB"),
4               ("left", "MULT", "DIV")
5           )
6
7
```

Capítulo 4

Exemplos de funcionamento

— preencher exemplos –

Capítulo 5

Conclusão

Fazendo uma retrospetiva referente ao trabalho prático, entendemos que os todos os objetivos do trabalho prático foram cumpridos. A realização deste trabalho foi particularmente atrativa pois ao desenvolver a nossa própria linguagem de programação somos nós quem decide toda a sua sintaxe e notação e chegar ao fim e perceber que conseguimos desenvolver a base de uma linguagem de programação é satisfatório.

A realização deste trabalho prático fez com que ficássemos bem dentro do funcionamento do módulo Lexer e do módulo Yacc, nomeadamente como funciona o reconhecimento de tokens e a implementação da nossa gramática. A geração de código Assembly foi sem dúvida também um ponto positivo deste trabalho pois permitiu-nos entender melhor a linguagem e as suas instruções.

Em suma, entendemos que todos os objetivos foram concluídos e consideramos que este trabalho foi bastante desafiador e uma excelente fonte de conhecimento para desafios futuros.