4 ESPECIFICAÇÃO DA LINGUAGEM PORTUGOL

A definição formal da linguagem foi feita com base nas definições e exemplos apresentados no início e nos exercícios resolvidos ao final dos capítulos do livro Fundamentos da Programação de Computadores (ASCENCIO; CAMPOS, 2007).

4.1 ESTRUTURA SEQUENCIAL

Um **programa sequencial** (sem sub-rotinas definidas pelo programador) escrito em Portugol apresenta a seguinte estrutura básica:

Quadro 13 - Estrutura básica de um programa sequencial em Portugol

```
ALGORITMO
declaracoes
bloco_de_comandos
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 16)

`declaracoes` corresponde às declarações de variáveis (ver seção 4.1.1) e **`bloco_de_comandos`** representa todos os comandos que compõem o programa em sequência. As sub-rotinas são declaradas após o algoritmo principal (ver seção 4.6).

4.1.1 Declaração de variáveis

As **variáveis globais** são declaradas após a palavra reservada **`DECLARE`**, sendo uma linha para cada tipo de variável. Variáveis de um mesmo **tipo de dado** podem ser declaradas em uma mesma linha, com seus identificadores separados por vírgula e seu tipo informado após a lista de identificadores (ASCENCIO; CAMPOS, 2007, p. 16).

Quadro 14 - Declaração de variáveis em Portugol

```
DECLARE
     x NUMERICO
     y, z LITERAL
     teste LOGICO
```

(ASCENCIO; CAMPOS, 2007, p. 16)

As regras para a formação de **identificadores** em Portugol são (ASCENCIO; CAMPOS, 2007, p. 9):

a) os identificadores podem conter números, letras maiúsculas, letras minúsculas e o caractere sublinhado;

b) o primeiro caractere deve ser uma letra ou o caractere sublinhado;

c) não são permitidos espaços em branco nem caracteres especiais, como **!**, **@**, **#**, **$**, **%**, **&**, **+**, **-**, dentre outros; e

d) os identificadores não podem coincidir com as palavras reservadas da linguagem de programação.

As **palavras reservadas** da linguagem Portugol são apresentadas a seguir.

Tabela 3 - Palavras reservadas da linguagem Portugol

ALGORITMO	**FACA**	**LITERAL**	**REGISTRO**
ATE	**FALSO**	**LOGICO**	**REPITA**
DECLARE	**FIM**	**NAO**	**RETORNE**
E	**FIM_ALGORITMO**	**NUMERICO**	**SE**
ENQUANTO	**FIM_SUBROTINA**	**OU**	**SENAO**
ENTAO	**INICIO**	**PARA**	**SUB-ROTINA**
ESCREVA	**LEIA**	**PASSO**	**VERDADEIRO**

(autoria própria)

Os identificadores definidos pelo usuário não podem coincidir com nenhuma dessas palavras, assim como não podem coincidir com os nomes das sub-rotinas pré-definidas pela linguagem, apresentadas na seção 4.1.7.

Por essas regras, são exemplos de identificadores válidos: **x**, **y**, **media**, **nota1**, **salario_atual**, **SalarioMinimo**, **NOME**. São exemplos de identificadores inválidos: **1anota** (por começar com número), **nota 1** (por conter um espaço em branco), **x-y** (por possuir um caractere especial), **algoritmo** (por coincidir com palavra reservada).

Os tipos de dados definidos para a linguagem Portugol são (ASCENCIO; CAMPOS, 2007, p. 8-9,16):

a) **numérico:** usado para variáveis que devem armazenar números, como **-23**, **-23.45**, **0**, **98**, **346.89**;

b) **lógico:** usado para variáveis que devem assumir apenas os valores **VERDADEIRO** ou **FALSO**; e

c) **literal:** usado para armazenar um ou mais caracteres (letras maiúsculas, minúsculas, números e caracteres especiais) em sequência.

Convencionou-se que a linguagem é **insensível à capitalização**, ou seja, não faz distinção entre letras maiúsculas e minúsculas (ASCENCIO; CAMPOS, 2007, p. 17). Portanto, o identificador **`NUM`** equivale ao identificador **`num`**. Apenas como recurso gráfico, neste documento as palavras reservadas são grafadas em maiúsculas, tal como no livro.

4.1.2 Comando de atribuição

O **comando de atribuição** é utilizado para armazenar valores ou resultados de operações em variáveis. Em Portugol, ele é representado pelo símbolo ← (ASCENCIO; CAMPOS, 2007, p. 16). Considerando as variáveis declaradas no exemplo do Quadro 14, alguns exemplos de comandos de atribuição são mostrados no Quadro 15.

Quadro 15 - Comando de atribuição em Portugol

```
x ← 4
x ← x + 2
y ← “aula”
teste ← FALSO
```

(ASCENCIO; CAMPOS, 2007, p. 16)

Na linguagem Portugol, as constantes do tipo literal são representadas entre aspas duplas e os números reais utilizam o ponto como separador decimal (ASCENCIO; CAMPOS, 2007, p. 8).

Como não é direto inserir o símbolo ← pelo teclado, escolheu-se representar o operador de atribuição por um sinal de menor (**`<`**) seguido por um traço (**`-`**).

4.1.3 Comando de entrada de dados

O **comando de entrada** é utilizado para instruir o computador a receber dados digitados pelo usuário e armazená-los nas variáveis. Em Portugol, o comando de entrada é iniciado pela palavra reservada **`LEIA`**, seguida de uma ou mais variáveis (nesse caso, separadas por vírgulas) cujos valores deseja-se obter do usuário (ASCENCIO; CAMPOS, 2007, p. 16).

Considerando as variáveis declaradas no exemplo do Quadro 14, para executar o comando mostrado no Quadro 16, o computador deve receber um número do usuário e armazená-lo na variável **x**. Um erro deve ser informado se o usuário não digitar um número.

Quadro 16 - Exemplo de comando de entrada de dados em Portugol

```
LEIA x
```

(ASCENCIO; CAMPOS, 2007, p. 16)

Ainda considerando aquelas variáveis, para executar o comando a seguir, o computador deve ler um ou vários caracteres digitados pelo usuário e armazená-los na variável **y**.

Quadro 17 - Outro exemplo de comando de entrada de dados em Portugol

```
LEIA y
```

(ASCENCIO; CAMPOS, 2007, p. 16)

4.1.4 Comando de saída de dados

O **comando de saída** é utilizado para instruir o computador a mostrar dados na tela. Em Portugol, o comando de saída é iniciado pela palavra reservada **ESCREVA**, seguida de uma ou mais expressões (nesse caso, separadas por vírgulas), que podem ser valores constantes ou variáveis (ASCENCIO; CAMPOS, 2007, p. 17).

Considerando as variáveis declaradas no exemplo do Quadro 14, para executar o comando a seguir, o computador deve mostrar na tela o número armazenado na variável **x**.

Quadro 18 - Exemplo de comando de saída de dados em Portugol

```
ESCREVA x
```

(ASCENCIO; CAMPOS, 2007, p. 17)

Ainda considerando aquelas variáveis, para executar o comando a seguir, o computador deve exibir na tela a mensagem “Conteúdo de y = ” e, logo em seguida, a sequência de caracteres armazenados na variável **y**.

Quadro 19 - Outro exemplo de comando de saída de dados em Portugol

```
ESCREVA "Conteúdo de y = ", y
```

(ASCENCIO; CAMPOS, 2007, p. 17)

Convencionou-se que cada comando de saída produz uma linha na tela. Assim, ao executar um comando de saída, o computador deve exibir na tela, sem mudar de linha, o valor de cada uma das expressões em sequência e, ao final da execução do comando, passar o cursor para a próxima linha.

4.1.5 Comentários

Comentários são textos que podem ser inseridos em códigos-fonte com o objetivo de documentá-los. Os comentários são ignorados pelo compilador durante a análise do código-fonte (ASCENCIO; CAMPOS, 2007, p. 24).

Ascencio e Campos (2007) não definem comentários para a linguagem Portugol. Por ser esse um recurso comumente encontrado nas linguagens de programação, decidiu-se permitir **comentários de linha**, em que a região de um comentário é iniciada por duas barras em sequência (`//`) e encerrada automaticamente ao final da linha, a semelhança dos comentários de linha definidos para as linguagems C e C++ (ASCENCIO; CAMPOS, 2007, p. 24). O Quadro 20 apresenta um exemplo de comentário válido na linguagem Portugol.

Quadro 20 - Comentário em Portugol

```
LEIA x // Obtém o valor de x do usuário
```

(autoria própria)

4.1.6 Operadores

Os operadores aritméticos definidos para a linguagem Portugol são mostrados na Tabela 4 e são aplicáveis somente aos dados numéricos.

Os operadores relacionais definidos para a linguagem Portugol são mostrados na Tabela 5 e são aplicáveis aos dados numéricos. Os operadores de igualdade e diferença são aplicáveis também a dados de mesmo tipo.

Os operadores lógicos definidos para a linguagem Portugol são mostrados na Tabela 6 e são aplicáveis somente aos dados lógicos.

Tabela 4 - Operadores aritméticos da linguagem Portugol

Operador	Exemplo	Descrição
`+`	`x + y`	Soma o valor de **x** e de **y**
`-`	`x - y`	Subtrai o valor de **x** do valor de **y**
`*`	`x * y`	Multiplica o valor de **x** pelo valor de **y**
`/`	`x / y`	Obtém o quociente da divisão de **x** por **y**
`+`	`+x`	Equivale a multiplicar **x** por `+1`
`-`	`-x`	Equivale a multiplicar **x** por `-1`

(autoria própria)

Tabela 5 - Operadores relacionais da linguagem Portugol

Operador	Exemplo	Descrição
`=`	`x = y`	O valor de **x** é igual ao valor de **y**
`<>`	`x <> y`	O valor de **x** é diferente do valor de **y**
`<`	`x < y`	O valor de **x** é menor que o valor de **y**
`<=`	`x <= y`	O valor de **x** é menor ou igual ao valor de **y**
`>`	`x > y`	O valor de **x** é maior que o valor de **y**
`>=`	`x >= y`	O valor de **x** é maior ou igual ao valor de **y**

(autoria própria)

Tabela 6 - Operadores lógicos da linguagem Portugol

Operador	Exemplo	Descrição
`OU`	`p OU q`	O valor de **p** ou o valor de **q**
`E`	`p E q`	O valor de **p** e o valor de **q**
`NAO`	`NAO p`	Nega o valor de **p**

(autoria própria)

Quando dois ou mais operadores aparecem em sequência em uma expressão, as operações são executadas na ordem estabelecida pelas convenções matemáticas (a multiplicação é executada antes da adição, por exemplo). As regras de precedência adotadas para os operadores são semelhantes às regras definidas para a linguagem C (MICROSOFT, 2015), comuns a muitas linguagens de programação.

As regras de precedência para a linguagem Portugol encontram-se resumidas na Tabela 7. Os operadores estão ordenados quanto à precedência de maneira crescente: os operadores situados nas linhas mais abaixo da tabela apresentam maior precedência que os operadores nas linhas mais acima; operadores em uma mesma linha apresentam a mesma precedência, sendo avaliados na sequência determinada pela sua associatividade.

Tabela 7 - Regras de precedência dos operadores da linguagem Portugol

Operadores	Associatividade
ou (disjunção)	Esquerda para direita
e (conjunção)	Esquerda para direita
= (igualdade) e **<>** (diferença)	Esquerda para direita
< (menor), **<=** (menor ou igual), **>** (maior) e **>=** (maior ou igual)	Esquerda para direita
+ (soma) e **-** (subtração)	Esquerda para direita
***** (multiplicação) e **/** (divisão)	Esquerda para direita
+ (positivo), **-** (negativo) e **nao** (negação)	Direita para esquerda

(autoria própria)

De maneira análoga à Matemática, a linguagem Portugol permite alterar a ordem em que as operações são executadas utilizando parênteses. Assim, a expressão `1 + 5 * 3`, por exemplo, resulta no valor **16**, enquanto a expressão `(1 + 5) * 3` resulta no valor **18**.

Um exemplo de programa completo escrito em Portugol é apresentado no Quadro 21.

Quadro 21 - Exemplo de programa em Portugol

```
// Faça um algoritmo para mostrar o resultado da multiplicação de dois números.

ALGORITMO
DECLARE n1, n2, m NUMERICO
ESCREVA "Digite dois números:"
LEIA n1, n2
m <- n1 * n2
ESCREVA "Multiplicação = ", m
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 4-5)

4.1.7 Sub-rotinas pré-definidas

A linguagem Portugol possui sub-rotinas predefinidas destinadas a cálculos matemáticos, apresentadas na Tabela 8.

Tabela 8 - Sub-rotinas predefinidas destinadas a cálculos da linguagem Portugol

Sub-rotina	Descrição	Exemplo

`arredonda(x)`	**Parâmetro:**
	• **x**: número real a ser arredondado
	Retorno:
	• o arredondamento do número real **x**
		`DECLARE i NUMERICO`
		`i <- arredonda(3.6)`
		`// Equivale a`
		`// i <- 4`

`cosseno(x)`	**Parâmetro:**
	• **x**: ângulo representado em radianos
	Retorno:
	• o cosseno do ângulo **x**
		`DECLARE ang, rad, cos NUMERICO`
		`ang <- 60`
		`rad <- ang * 3.14 / 180`
		`cos <- cosseno(rad)`
		`// Equivale a`
		`// cos <- 0.5`

`parte_inteira(x)`	**Parâmetro:**
	• **x:** número real do qual se deseja obter a parte inteira
	Retorno:
	• a parte inteira do número real **x**
		`DECLARE i NUMERICO`
		`i <- parte_inteira(3.6)`
		`// Equivale a`
		`// i <- 3`

`potencia(a, b)`	**Parâmetros:**
	• **a:** a base
	• **b:** o expoente
	Retorno:
	• o número **a** elevado ao número **b**
		`DECLARE p NUMERICO`
		`p <- potencia(2, 10)`
		`// Equivale a`
		`// p <- 1024`

`raiz_enesima(n, x)`	**Parâmetros:**
	• **n:** índice da raiz
	• **x:** número do qual se deseja obter a raiz **n**
	Retorno:
	• a raiz **n** do número **x**
		`DECLARE r3 NUMERICO`
		`r3 <- raiz_enesima(3, 8)`
		`// Equivale a`
		`// r3 <- 2`

`raiz_quadrada(x)`	**Parâmetro:**
	• **x:** número do qual se deseja obter a raiz quadrada
	Retorno:
	• a raiz quadrada do número **x**
		`DECLARE r2 NUMERICO`
		`r2 <- raiz_quadrada(4)`
		`// Equivale a`
		`// r2 <- 2`

`resto(x, y)`	**Parâmetros:**
	• **x:** dividendo
	• **y:** divisor
	Retorno:
	• o resto da divisão do número **x** pelo número **y**
		`DECLARE r NUMERICO`
		`r <- resto(8, 3)`
		`// Equivale a`
		`// r <- 2`

`seno(x)`	**Parâmetro:**
	• **x:** ângulo representado em radianos
	Retorno:
	• o seno do ângulo **x**
		`DECLARE ang, rad, sen NUMERICO`
		`ang <- 30`
		`rad <- ang * 3.14 / 180`
		`sen <- seno(rad)`
		`// Equivale a`
		`// sen <- 0.5`

(autoria própria)

Além dessas, foram definidas outras sub-rotinas, apresentadas na Tabela 9.

Tabela 9 - Outras sub-rotinas predefinidas da linguagem Portugol

Sub-rotina	Descrição	Exemplo
`limpar_tela()`	apaga todos os caracteres da tela	`limpar_tela()`

`obtenha_ano(data)`	**Parâmetro:**
	• **`data`:** data obtida pela sub-rotina **`obtenha_data()`**
	Retorno:
	• o ano da **`data`** fornecida
		`DECLARE hoje, ano NUMERICO`
		`hoje <- obtenha_data()`
		`ano <- obtenha_ano(hoje)`

`obtenha_data()`	**Retorno:** • a diferença, medida em milissegundos, entre a data atual e 01/01/1970	`DECLARE hoje NUMERICO` `hoje <- obtenha_data()`

`obtenha_dia(data)`	**Parâmetro:**
	• **`data`:** data obtida pela sub-rotina **`obtenha_data()`**
	Retorno:
	• o dia da **`data`** fornecida
		`DECLARE hoje, dia NUMERICO`
		`hoje <- obtenha_data()`
		`dia <- obtenha_dia(hoje)`

`obtenha_hora(horario)`	**Parâmetro:**
	• **`horario`:** horário obtido pela sub-rotina **`obtenha_horario()`**
	Retorno:
	• as horas do **`horario`** fornecido
		`DECLARE agora, hora NUMERICO`
		`agora <- obtenha_horario()`
		`hora <- obtenha_hora(agora)`

`obtenha_horario()`	**Retorno:** • a diferença, medida em milissegundos, entre a data e hora atuais e 01/01/1970 00:00	`DECLARE agora NUMERICO` `agora <- obtenha_horario()`

`obtenha_mes(data)`	**Parâmetro:**
	• **`data`:** data obtida pela sub-rotina **`obtenha_data()`**
	Retorno:
	• o mês da **`data`** fornecida
		`DECLARE hoje, mes NUMERICO`
		`hoje <- obtenha_data()`
		`mes <- obtenha_mes(hoje)`

`obtenha_minuto(horario)`	**Parâmetro:**
	• **`horario`:** horário obtido pela sub-rotina **`obtenha_horario()`**
	Retorno:
	• os minutos do **`horario`** fornecido
		`DECLARE agora, minuto NUMERICO`
		`agora <- obtenha_horario()`
		`minuto <- obtenha_minuto(agora)`

`obtenha_segundo(horario)`	**Parâmetro:**
	• **`horario`:** horário obtido pela sub-rotina **`obtenha_horario()`**
	Retorno:
	• os segundos do **`horario`** fornecido
		`DECLARE agora, segundo NUMERICO`
		`agora <- obtenha_horario()`
		`segundo <- obtenha_segundo(agora)`

(autoria própria)

A linguagem Portugol não permite a manipulação de cadeias de caracteres nem o

manuseio de arquivos. Ascencio e Campos (2007) em nenhum momento definem operações como, por exemplo, concatenação de cadeias de caracteres ou escrita em arquivo para a linguagem Portugol.

4.2 ESTRUTURAS CONDICIONAIS

A **estrutura condicional** permite estabelecer que um comando ou bloco de comandos deve ser executado somente se determinada condição for verdadeira. Opcionalmente, pode-se especificar também um comando ou bloco de comandos para ser executado se a condição for falsa. O primeiro tipo de estrutura condicional é chamado de **estrutura condicional simples**, e o segundo, **estrutura condicional composta** (ASCENCIO; CAMPOS, 2007, p. 50).

4.2.1 Estrutura condicional simples

A estrutura condicional simples em Portugol apresenta a seguinte estrutura básica:

Quadro 22 - Estrutura condicional simples em Portugol

```
SE condicao ENTAO
     comando
```

(ASCENCIO; CAMPOS, 2007, p. 50)

comando é um comando que deve ser executado apenas se **condicao**, que pode ser uma variável ou expressão do tipo lógico, apresentar o valor **VERDADEIRO**. No lugar de apenas um comando, é possível especificar um bloco de comandos, delimitando-o com as palavras reservadas **INICIO** e **FIM** (ASCENCIO; CAMPOS, 2007, p. 50):

Quadro 23 - Estrutura condicional simples utilizando blocos de comandos em Portugol

```
SE condicao ENTAO
INICIO
     comando_1
     comando_2
     …
     comando_m
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 50)

Nesse caso, os comandos contidos no bloco de comandos serão executados em sequência caso **condicao** apresente o valor **VERDADEIRO**.

Um exemplo de programa que utiliza a estrutura condicional simples é apresentado no Quadro 24.

Quadro 24 - Exemplo de uso de estrutura condicional simples em Portugol

```
// Faça um programa que receba dois números e mostre o maior.

ALGORITMO
DECLARE num1, num2 NUMERICO
ESCREVA "Digite o primeiro número:"
LEIA num1
ESCREVA "Digite o segundo número:"
LEIA num2
SE num1 > num2 ENTAO
     ESCREVA "O maior número é: ", num1
SE num2 > num1 ENTAO
     ESCREVA "O maior número é: ", num2
SE num1 = num2 ENTAO
     ESCREVA "Os números são iguais"
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 60)

4.2.2 Estrutura condicional composta

A estrutura condicional composta em Portugol apresenta a seguinte estrutura básica:

Quadro 25 - Estrutura condicional composta em Portugol

```
SE condicao ENTAO
     comando_1
SENAO
     comando_2
```

(ASCENCIO; CAMPOS, 2007, p. 50)

comando_1 é um comando que deve ser executado apenas se **condicao**, que pode ser uma variável ou expressão do tipo lógico, apresentar o valor **VERDADEIRO**, e **comando_2** é um comando que deve ser executado caso contrário. No lugar de apenas um comando, é possível especificar um bloco de comandos, de maneira análoga à estrutura condicional simples, como é mostrado no Quadro 26.

Um exemplo de programa que utiliza a estrutura condicional composta é apresentado no Quadro 27.

Quadro 26 - Estrutura condicional composta utilizando blocos de comandos em Portugol

```
SE condicao ENTAO
INICIO
     comando_1
     comando_2
     …
     comando_m
FIM
SENAO
INICIO
     comando_1
     comando_2
     …
     comando_m
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 50)

Quadro 27 - Estrutura condicional composta utilizando blocos de comandos em Portugol

```
// Faça um programa que receba um número inteiro e verifique se é par ou
// ímpar.

ALGORITMO
DECLARE num, r NUMERICO
ESCREVA "Digite um número:"
LEIA num
r <- resto(num, 2)
SE r = 0 ENTAO
     ESCREVA "O número é par"
SENAO
     ESCREVA "O número é ímpar"
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 62)

4.3 ESTRUTURAS DE REPETIÇÃO

A **estrutura de repetição** permite estabelecer que um comando ou bloco de comandos deve ser executado repetidas vezes. O número de repetições pode ser fixado ou vinculado à verificação de uma condição. Para possibilitar que a repetição seja definida da forma mais conveniente a quem desenvolve o programa, a linguagem Portugol possui três tipos de estruturas de repetição (ASCENCIO; CAMPOS, 2007, p. 93), apresentadas a seguir.

4.3.1 Estrutura de repetição PARA

A estrutura de repetição **PARA** é utilizada quando se sabe o número de vezes que a repetição deve ocorrer. Ela apresenta o seguinte formato:

Quadro 28 - Estrutura de repetição PARA em Portugol

```
PARA indice <- valor_inicial ATE valor_final FACA [PASSO n]
     comando
```

(ASCENCIO; CAMPOS, 2007, p. 93)

comando é um comando que deve ser executado para **indice** variando de **valor_inicial** até **valor_final**. Opcionalmente, pode-se definir o valor em que a variável **indice** é incrementada (ou decrementada), fornecendo um valor inteiro **n** após a palavra reservada **PASSO**, inserida após a palavra reservada **FACA**. Por padrão, se **n** não for fornecido, o valor da variável **indice** será incrementado de 1 unidade após cada repetição. No lugar de apenas um comando, é possível especificar um bloco de comandos a serem repetidos:

Quadro 29 - Estrutura de repetição PARA com vários comandos em Portugol

```
PARA indice <- valor_inicial ATE valor_final FACA [PASSO n]
INICIO
     comando_1
     comando_2
     …
     comando_m
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 93)

Um exemplo de programa que utiliza a estrutura de repetição **PARA** é apresentado no Quadro 30.

4.3.2 Estrutura de repetição ENQUANTO

A estrutura de repetição **ENQUANTO** é utilizada quando não se sabe o número de vezes que um comando (ou um bloco de comandos) deve ser executado e possivelmente ele não será executado, por isso uma condição deve ser testada antes que cada iteração da repetição seja executada (ASCENCIO; CAMPOS, 2007, p. 94).

Quadro 30 - Exemplo de uso de estrutura de repetição PARA em Portugol

```
// Faça um programa que leia um valor N inteiro e positivo, calcule e
// mostre o valor de E, conforme a fórmula a seguir:
//
// E = 1 + 1/1! + 1/2! + 1/3! + ... + 1/N!

ALGORITMO
DECLARE n, euler, i, j, fat NUMERICO
LEIA n
euler <- 1
PARA i <- 1 ATE n FACA
INICIO
     fat <- 1
     PARA j <- 1 ATE i FACA
     INICIO
          fat <- fat * j
     FIM
     euler <- euler + 1/fat
FIM
ESCREVA euler
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 109)

A estrutura de repetição **ENQUANTO** apresenta o seguinte formato:

Quadro 31 - Estrutura de repetição ENQUANTO em Portugol

```
ENQUANTO condicao FACA
     comando
```

(ASCENCIO; CAMPOS, 2007, p. 94)

comando é um comando que deve ser executado enquanto **condicao**, que pode ser uma variável ou expressão do tipo lógico, apresentar o valor **VERDADEIRO**. No lugar de apenas um comando, é possível especificar um bloco de comandos a serem repetidos:

Quadro 32 - Estrutura de repetição ENQUANTO com vários comandos em Portugol

```
ENQUANTO condicao FACA
INICIO
     comando_1
     comando_2
     …
     comando_m
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 94)

Um exemplo de programa que utiliza a estrutura de repetição **ENQUANTO** é apresentado no Quadro 33.

Quadro 33 - Exemplo de uso de estrutura de repetição ENQUANTO em Portugol

```
// Faça um programa que leia um conjunto não determinado de valores, um de
// cada vez, e escreva uma tabela com cabeçalho, que deve ser repetido a
// cada vinte linhas. A tabela deverá conter o valor lido, seu quadrado,
// seu cubo e sua raiz quadrada. Finalize a entrada de dados com um valor
// negativo ou zero.

ALGORITMO
DECLARE linhas, num, quad, cubo, raiz NUMERICO
LEIA num
ESCREVA "Valor - Quadrado - Cubo - Raiz"
linhas <- 1
ENQUANTO num > 0 FACA
INICIO
     quad <- num * num
     cubo <- num * num * num
     raiz <- raiz_quadrada(num)
     SE linhas < 20 ENTAO
     INICIO
          linhas <- linhas + 1
          ESCREVA num, " - ", quad, " - ", cubo, " - ", raiz
     FIM
     SENAO
     INICIO
          limpar_tela()
          linhas <- 1
          ESCREVA "Valor - Quadrado - Cubo - Raiz"
          linhas <- linhas + 1
          ESCREVA num, " - ", quad, " - ", cubo, " - ", raiz
     FIM
     LEIA num
FIM
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 129)

4.3.3 Estrutura de repetição REPITA

A estrutura de repetição **REPITA** é utilizada quando não se sabe o número de vezes que um comando (ou um bloco de comandos) deve ser executado, mas a execução deve ocorrer pelo menos uma vez, por isso uma condição deve ser testada após cada iteração para verificar se a repetição deve ocorrer uma vez mais (ASCENCIO; CAMPOS, 2007, p. 95).

A estrutura de repetição **REPITA** apresenta o seguinte formato:

Quadro 34 - Estrutura de repetição REPITA em Portugol

```
REPITA
     comando
ATE condicao
```

(ASCENCIO; CAMPOS, 2007, p. 95)

comando é um comando que deve ser executado até que **condicao**, que pode ser uma variável ou expressão do tipo lógico, apresente o valor **VERDADEIRO**.

No lugar de apenas um comando, é possível especificar um conjunto de comandos a serem repetidos:

Quadro 35 - Estrutura de repetição REPITA com vários comandos em Portugol

```
REPITA
     comando_1
     comando_2
     …
     comando_m
ATE condicao
```

(ASCENCIO; CAMPOS, 2007, p. 95)

Um exemplo de programa que utiliza a estrutura de repetição **REPITA** é apresentado no Quadro 36.

Quadro 36 - Exemplo de uso de estrutura de repetição REPITA em Portugol

```
// Faça um programa que conte regressivamente de 10 até 1.

ALGORITMO
DECLARE contador NUMERICO
contador <- 10
REPITA
      ESCREVA contador
      contador <- contador - 1
ATE contador = 0
ESCREVA "Fim!"
FIM_ALGORITMO.
```

(autoria própria)

4.4 VETORES

Um **vetor**, também conhecido por **variável composta homogênea unidimensional**, é um conjunto de posições de memória que armazenam o mesmo tipo de dado. Elas possuem o mesmo identificador, são armazenadas sequencialmente na memória e se distinguem por um índice, que indica sua localização dentro da estrutura (ASCENCIO; CAMPOS, 2007, p. 145).

4.4.1 Declaração de vetores

Os vetores são declarados após a palavra reservada **`DECLARE`**, no mesmo lugar destinado à declaração de variáveis, sendo uma linha para cada tipo de dado (ASCENCIO; CAMPOS, 2007, p. 145), como é mostrado o Quadro 37.

Quadro 37 - Declaração de vetores em Portugol

```
DECLARE nome[tamanho] tipo
```

(ASCENCIO; CAMPOS, 2007, p. 145)

`nome` é o identificador do vetor, **`tamanho`** é a quantidade de variáveis que devem compor o vetor e **`tipo`** é o tipo dos dados que serão armazenados no vetor (numérico, lógico ou literal, como visto na seção 4.1.1, ou registro, como visto na seção 4.7).

Seja, por exemplo, a seguinte declaração de vetor:

Quadro 38 - Exemplo de declaração de vetor em Portugol

```
DECLARE x[5] NUMERICO
```

(ASCENCIO; CAMPOS, 2007, p. 145)

Ela indica que 5 posições de memória devem ser reservadas em sequência para o armazenamento de 5 números.

4.4.2 Uso de vetores

O acesso a uma posição de memória em um vetor exige que sua localização no vetor seja indicada após o identificador do vetor (ASCENCIO; CAMPOS, 2007, p. 145). Esse acesso pode ser feito de qualquer lugar onde é possível acessar uma variável.

Considerando a declaração de vetor exemplificada no Quadro 38, o comando de atribuição a seguir, por exemplo, indica que o valor **45** deve ser armazenado na posição **1** do vetor **x**.

Quadro 39 - Uso de vetores em Portugol

```
x[1] <- 45
```

(ASCENCIO; CAMPOS, 2007, p. 145)

Para preencher um vetor, uma estrutura de repetição **PARA** pode ser utilizada em conjunto com o comando de entrada de dados, como no exemplo a seguir.

Quadro 40 - Exemplo de preenchimento de vetor em Portugol

```
PARA i <- 1 ATE 5 FACA
INICIO
     ESCREVA "Digite o ", i, "º número"
     LEIA x[i]
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 145-146)

De maneira semelhante, para exibir todos os valores armazenados em um vetor, uma estrutura de repetição **PARA** pode ser utilizada em conjunto com o comando de saída de dados, como no exemplo a seguir.

Quadro 41 - Exemplo de exibição de vetor em Portugol

```
PARA i <- 1 ATE 5 FACA
INICIO
     ESCREVA "Este é o ", i, "º número"
     ESCREVA x[i]
FIM
```

(ASCENCIO; CAMPOS, 2007, p. 146)

Um exemplo de programa que utiliza vetores é apresentado no Quadro 42.

Quadro 42 - Exemplo de uso de vetores em Portugol

```
// Faça um programa que preencha um vetor com nove números inteiros,
// calcule e mostre os números primos e suas respectivas posições.

ALGORITMO
DECLARE
     num[9] NUMERICO
     i, j, cont NUMERICO
PARA i <- 1 ATE 9 FACA
INICIO
     LEIA num[i]
FIM
PARA i <- 1 ATE 9 FACA
INICIO
     cont <- 0
     PARA j <- 1 ATE num[i] FACA
     INICIO
          SE resto(num[i], j) = 0 ENTAO
               cont <- cont + 1
     FIM
     SE cont <= 2 ENTAO
     INICIO
          ESCREVA num[i], " - ", i
     FIM
FIM
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 151)

4.5 MATRIZES

Uma **matriz**, também conhecida por **variável composta homogênea multidimensional**, é um conjunto de posições de memória que armazenam o mesmo tipo de dado. Elas possuem o mesmo identificador, são armazenadas sequencialmente na memória e se distinguem por índices, que indicam sua localização dentro da estrutura, sendo um índice para cada uma das dimensões da matriz (ASCENCIO; CAMPOS, 2007, p. 187).

4.5.1 Declaração de matrizes

As matrizes são declaradas após a palavra reservada **`DECLARE`**, no mesmo lugar destinado à declaração de variáveis e vetores, sendo uma linha para cada tipo de dado (ASCENCIO; CAMPOS, 2007, p. 187), como é mostrado no Quadro 43.

Quadro 43 - Declaração de matrizes em Portugol

```
DECLARE nome[dimensao_1, dimensao_2, …, dimensao_n] tipo
```

(ASCENCIO; CAMPOS, 2007, p. 187)

nome é o identificador da matriz, **dimensao_1**, **dimensao_2**, …, **dimensao_n** indicam os tamanhos das dimensões 1, 2, …, n da matriz, respectivamente, e **tipo** é o tipo dos dados que serão armazenados na matriz (numérico, lógico ou literal, como visto na seção 4.1.1, ou registro, como visto na seção 4.7).

Seja, por exemplo, a declaração de matriz apresentada no Quadro 44.

Quadro 44 - Exemplo de declaração de matriz em Portugol

```
DECLARE x[3,5] NUMERICO
```

(ASCENCIO; CAMPOS, 2007, p. 187)

Essa declaração define uma matriz bidimensional (análoga a uma tabela) em que o tamanho da primeira dimensão (linha) é 3 e o da segunda dimensão (coluna) é 5.

4.5.2 Uso de matrizes

O acesso a uma posição de memória em uma matriz exige que sua localização na matriz seja indicada após o identificador da matriz (ASCENCIO; CAMPOS, 2007, p. 188). Esse acesso pode ser feito de qualquer lugar onde é possível acessar uma variável ou vetor.

Considerando a declaração de matriz exemplificada no Quadro 44, o comando de atribuição a seguir, por exemplo, indica que o valor **13** deve ser armazenado na linha **3**, coluna **1** da matriz bidimensional **x**.

Quadro 45 - Uso de matrizes em Portugol

```
x[3,1] <- 13
```

(ASCENCIO; CAMPOS, 2007, p. 188)

Para preencher uma matriz, estruturas de repetição **PARA** (uma para cada dimensão) podem ser utilizadas em conjunto com o comando de entrada de dados, como no exemplo a seguir.

Quadro 46 - Exemplo de preenchimento de matriz em Portugol

```
PARA i <- 1 ATE 3 FACA
     PARA j <- 1 ATE 5 FACA
     INICIO
          ESCREVA "Digite o número da linha ", i, " e coluna ", j
          LEIA x[i,j]
     FIM
```

(ASCENCIO; CAMPOS, 2007, p. 188)

De maneira semelhante, para exibir todos os valores armazenados em uma matriz, estruturas de repetição **PARA** podem ser utilizadas em conjunto com o comando de saída de dados, como no exemplo a seguir.

Quadro 47 - Exemplo de exibição de matriz em Portugol

```
PARA i <- 1 ATE 3 FACA
     PARA j <- 1 ATE 5 FACA
     INICIO
          ESCREVA "Este é o número da linha ", i, " e coluna ", j
          ESCREVA x[i,j]
     FIM
```

(ASCENCIO; CAMPOS, 2007, p. 190)

Um exemplo de programa que utiliza matrizes é apresentado no Quadro 48.

4.6 SUB-ROTINAS

Sub-rotinas, também chamadas de subprogramas, são blocos de instruções que realizam tarefas específicas. Elas permitem subdividir o problema a ser resolvido pelo programa em problemas menores. As sub-rotinas podem ou não receber argumentos, assim como podem ou não retornar um valor (ASCENCIO; CAMPOS, 2007, p. 230).

Dentro das sub-rotinas pode ocorrer declaração de variáveis, chamadas **variáveis locais** porque podem ser utilizadas apenas dentro da sub-rotina e perdem seu conteúdo quando a execução da sub-rotina termina. Variáveis declaradas no algoritmo são chamadas **variáveis globais** porque podem ser acessadas em qualquer ponto do programa e são destruídas apenas quando a execução deste termina (ASCENCIO; CAMPOS, 2007, p. 231).

Quadro 48 - Exemplo de uso de matrizes em Portugol

```
// Faça um programa que preencha uma matriz M(2x2), calcule e mostre a
// matriz R, resultante da multiplicação dos elementos de M pelo seu maior
// elemento.

ALGORITMO
DECLARE mat[2,2], resultado[2,2], i, j, maior NUMERICO
PARA i <- 1 ATE 2 FACA
INICIO
     PARA j <- 1 ATE 2 FACA
     INICIO
          LEIA mat[i,j]
     FIM
FIM
maior <- mat[1,1]
PARA i <- 1 ATE 2 FACA
INICIO
     PARA j <- 1 ATE 2 FACA
     INICIO
          SE mat[i,j] > maior ENTAO
               maior <- mat[i,j]
     FIM
FIM
PARA i <- 1 ATE 2 FACA
INICIO
     PARA j <- 1 ATE 2 FACA
     INICIO
          resultado[i,j] <- maior * mat[i,j]
     FIM
FIM
PARA i <- 1 ATE 2 FACA
INICIO
     PARA j <- 1 ate 2 FACA
     INICIO
          ESCREVA resultado[i,j]
     FIM
FIM
FIM_ALGORITMO.
```

(ASCENCIO; CAMPOS, 2007, p. 198)

4.6.1 Passagem de parâmetros

A passagem de parâmetros ocorre por **referência**, ou seja, os parâmetros passados para a sub-rotina correspondem aos endereços de memória representados pelas variáveis. Assim, é possível alterar o valor do parâmetro durante a execução da sub-rotina e perceber essa alteração na volta para a execução do algoritmo principal (ASCENCIO; CAMPOS, 2007, p. 239). Um exemplo de programa que utiliza passagem de parâmetros por referência é apresentado no Quadro 49.

Quadro 49 - Exemplo de passagem de parâmetros por referência em Portugol

```
// Crie um programa que carregue uma matriz 3x4 com números reais. Utilize
// uma função para copiar todos os valores da matriz para um vetor de doze
// posições. Este vetor deverá ser mostrado no programa principal.

ALGORITMO
DECLARE mat[3,4], vet[12], i, j NUMERICO
PARA i <- 1 ATE 3 FACA
INICIO
     PARA j <- 1 ATE 4 FACA
     INICIO
          LEIA mat[i,j]
     FIM
FIM
gera_vetor(mat, vet)
PARA i <- 1 ATE 12 FACA
INICIO
     ESCREVA vet[i]
FIM
FIM_ALGORITMO

SUB-ROTINA gera_vetor(m[3,4], v[12] NUMERICO)
DECLARE i, j, k NUMERICO
k <- 1
PARA i <- 1 ATE 3 FACA
INICIO
     PARA j <- 1 ATE 4 FACA
     INICIO
          v[k] <- m[i,j]
          k <- k + 1
     FIM
FIM
FIM_SUB_ROTINA gera_vetor
```

(ASCENCIO; CAMPOS, 2007, p. 265)

4.6.2 Retorno

Para encerrar a execução de uma sub-rotina e retornar à execução do algoritmo principal deve-se utilizar o comando **RETORNE**, fornecendo-lhe o valor a ser retornado (ASCENCIO; CAMPOS, 2007, p. 230).

Um exemplo de programa que utiliza uma sub-rotina que retorna valor é apresentado no Quadro 50.

Quadro 50 - Exemplo de uso de sub-rotina que retorna valor em Portugol

```
// Faça um programa contendo uma sub-rotina que retorne 1 se o número
// digitado for positivo ou 0 se for negativo.

ALGORITMO
DECLARE num, x NUMERICO
LEIA num
x <- verifica(num)
SE x = 1 ENTAO
     ESCREVA "Número positivo"
SENAO
     ESCREVA "Número negativo"
FIM_ALGORITMO

SUB-ROTINA verifica(num NUMERICO)
DECLARE res NUMERICO
SE num >= 0 ENTAO
     res <- 1
SENAO
     res <- 0
RETORNE res
FIM_SUB_ROTINA verifica
```

(ASCENCIO; CAMPOS, 2007, p. 246)

4.7 REGISTROS

Um **registro**, também conhecido por **variável composta heterogênea**, é uma estrutura de dados capaz de agregar informações. Cada informação contida em um registro é chamada de **campo**. Os campos de um registro podem ser de diferentes tipos primitivos, vetores, matrizes, ou até mesmo registros (ASCENCIO; CAMPOS, 2007, p. 303).

4.7.1 Declaração de registros

Os registros são declarados após a palavra reservada **`DECLARE`**, no mesmo lugar destinado à declaração de variáveis, vetores e matrizes. Para cada registro, deve-se definir nomes e tipos de dados para seus campos (ASCENCIO; CAMPOS, 2007, p. 303):

Quadro 51 - Declaração de registros em Portugol

```
DECLARE nome REGISTRO (nome_do_campo_1 tipo_do_campo_1, nome_do_campo_2
tipo_do_campo_2, …, nome_do_campo_n tipo_do_campo_n)
```

(ASCENCIO; CAMPOS, 2007, p. 303)

O Quadro 52 mostra um exemplo de declaração de registro.

Também é possível declarar vetores ou matrizes de registros, como no exemplo do Quadro 53. Nesse caso, em cada posição de memória do vetor ou matriz será armazenado um registro com os campos definidos.

Quadro 52 - Exemplo de declaração de registro em Portugol

```
DECLARE conta registro (num, saldo NUMERICO nome LITERAL)
```

(ASCENCIO; CAMPOS, 2007, p. 303)

Quadro 53 - Exemplo de declaração de registro em Portugol

```
DECLARE conta[3] registro (num, saldo NUMERICO nome LITERAL)
```

(ASCENCIO; CAMPOS, 2007, p. 303)

4.7.2 Uso de registros

Para acessar um campo de um registro, é necessário indicar o nome da variável e o nome do campo desejado, separados por um ponto (ASCENCIO; CAMPOS, 2007, p. 303).

Considerando a declaração de registro exemplificada no Quadro 52, o comando de atribuição a seguir, por exemplo, indica que o valor **12** deve ser armazenado no campo **num** do registro **conta**.

Quadro 54 - Uso de registros em Portugol

```
conta.num <- 12
```

(ASCENCIO; CAMPOS, 2007, p. 304)

Considerando a declaração de vetor de registro exemplificada no Quadro 53, o comando de saída a seguir, por exemplo, indica que o valor armazenado no campo **num** do registro na posição **i** do vetor **conta** deve ser exibido para o usuário.

Quadro 55 - Uso de vetor de registros em Portugol

```
ESCREVA conta[i].num
```

(autoria própria)

Um exemplo de programa que utiliza um vetor de registros é mostrado no Quadro 56.

Quadro 56 - Exemplo de uso de vetor de registros em Portugol

```
// Faça um programa que realize o cadastro de contas bancárias com as
// seguintes informações: número da conta, nome do cliente e saldo. O
// banco permitirá o cadastramento de apenas quinze contas e não poderá
// haver mais que uma conta com o mesmo número. Crie o menu de opções a
// seguir.
//
// Menu de opções:
// 1. Cadastrar contas.
// 2. Visualizar todas as contas de determinado cliente.
// 3. Excluir a conta com menor saldo (supondo a não-existência de saldos
// iguais).
// 4. Sair.

ALGORITMO
DECLARE
     conta[15] REGISTRO (num, saldo NUMERICO nome LITERAL)
     i, op, posi, achou, num_conta, menor_saldo NUMERICO
     nome_cliente LITERAL
PARA i <- 1 ATE 15 FACA
INICIO
     conta[i].num <- 0
     conta[i].nome <- ""
     conta[i].saldo <- 0
FIM
posi <- 1
REPITA
     ESCREVA "Menu de Opções"
     ESCREVA "1. Cadastrar contas"
     ESCREVA "2. Visualizar todas as contas de determinado cliente"
     ESCREVA "3. Excluir a conta com menor saldo"
     ESCREVA "4. Sair"
     ESCREVA "Digite sua opção:"
     LEIA op
     SE op < 1 OU op > 4 ENTAO
          ESCREVA "Opção inválida!"
     SE op = 1 ENTAO
     INICIO
          SE posi > 15 ENTAO
               ESCREVA "Todas as contas já foram cadastradas!"
          SENAO
          INICIO
               achou <- 0
               ESCREVA "Digite o número da conta a ser incluída:"
               LEIA num_conta
               PARA i <- 1 ATE posi - 1 FACA
               INICIO
                    SE num_conta = conta[i].num ENTAO
                         achou <- 1
               FIM
               SE achou = 1 ENTAO
                    ESCREVA "Já existe conta cadastrada com esse número!"
               SENAO
               INICIO
```

```
                    conta[posi].num <- num_conta
                    ESCREVA "Digite o nome do cliente:"
                    LEIA conta[posi].nome
                    ESCREVA "Digite o saldo do cliente:"
                    LEIA conta[posi].saldo
                    ESCREVA "Conta cadastrada com sucesso:"
                    posi <- posi + 1
               FIM
          FIM
     FIM
     SE op = 2 ENTAO
     INICIO
          ESCREVA "Digite o nome do cliente a ser consultado:"
          LEIA nome_cliente
          achou <- 0
          PARA i <- 1 ATE posi - 1 FACA
          INICIO
               SE conta[i].nome = nome_cliente ENTAO
               INICIO
                    ESCREVA conta[i].num, " - ", conta[i].saldo
                    achou <- 1
               FIM
          FIM
          SE achou = 0 ENTAO
               ESCREVA "Não existe conta cadastrada para este cliente!"
     FIM
     SE op = 3 ENTAO
     INICIO
          SE posi = 1 ENTAO
               ESCREVA "Nenhuma conta foi cadastrada!"
          SENAO
          INICIO
               menor_saldo <- conta[1].saldo
               achou <- 1
               i <- 2
               ENQUANTO i < posi FACA
               INICIO
                    SE conta[i].saldo < menor_saldo ENTAO
                    INICIO
                         menor_saldo <- conta[i].saldo
                         achou <- i
                    FIM
                    i <- i + 1
               FIM
               SE achou <> posi - 1 ENTAO
                    PARA i <- achou + 1 ATE posi - 1 FACA
                    INICIO
                         conta[i - 1].num <- conta[i].num
                         conta[i - 1].nome <- conta[i].nome
                         conta[i - 1].saldo <- conta[i].saldo
                    FIM
               ESCREVA "Conta excluída com sucesso!"
               posi <- posi - 1
          FIM
     FIM
ATE op = 4
FIM_ALGORITMO.
```


(ASCENCIO; CAMPOS, 2007, p. 313-314)