# 2. PARADIGMA IMPERATIVO

A arquitetura dos computadores execeu um efeito crucial sobre o projeto das linguagens de programação. A maioria das linguagens populares nos últimos 35 anos foram projetadas em torno da arquitetura do computador, conhecidas por arquitetura de *von Neumann*. Estas linguagens são chamadas de imperativas. Em um computador com esta arquitetura, tanto os dados como os programas são armazenados na mesma memória, e a CPU (Unidade Central de Processamento), que executa realmente as instruções, é separada da memória. Consequentemente, instruções e dados devem ser transportados, ou transmitidos, da memória para a CPU. Os resultados das operações realizadas na CPU devem ser devolvidos para a memória.

Devido a arquitetura de *von Neumann*, as características centrais das linguagens imperativas são: as variáveis, que modelam as células de memória; comandos de atribuição, que são baseados nas operações de transferência dos dados e instruções; a execução seqüencial de instruções; e a forma iterativa de repetição, que é o método mais eficiente desta arquitetura. Os operandos das expressões são passados da memória para a CPU, e o resultado da expressão é passado de volta para a célula de memória, representada pelo lado esquerdo do comando de atribuição. A iteração é rápida em computadores com este tipo de arquitetura porque as instruções são armazenadas em células adjacentes da memória. Essa eficiência desencoraja o uso da recursão para repetição [SEB 99].

É interessante comentar que algumas vezes as linguagens de programação imperativas são também chamadas de procedurais, mas isto não tem relação com o conceito de procedimento. Neste capítulo, então, o paradigma imperativo de linguagens de programação, encontrado, por exemplo, nas linguagens *Fortran*, *Cobol*, *Basic*, *Pascal*, *Modula-2*, *C* e *Ada*, será apresentado. Inicialmente, na seção 2.1, é feita uma introdução ao assunto. Na seção 2.2 características da programação imperativa, bem como exemplos de LP que utilizam este paradigma, são descritas.

## 2.1. Introdução

O modelo imperativo de programação, que é o mais antigo de todos, baseia-se no modo de funcionamento do computador. Isto é refletido na execução seqüencial baseada em comandos e no armazenamento de dados alterável, conceitos que são baseados na maneira pela qual computadores executam programas a nível de linguagem de máquina ("*imperare*" em Latim significa "comandar"). O paradigma imperativo foi predominante nas LP, pois tais linguagens são mais fáceis de traduzir para uma forma adequada para execução na máquina. Um programa desenvolvido a partir deste modelo, por exemplo nas linguagens *C* e *Modula-2*, consiste em uma seqüência de modificações no armazenamento do computador [DER 90, WAT 90].

Conforme descrito anteriormente, linguagens imperativas são caracterizadas por três conceitos: variáveis, atribuições e seqüência. O estado de um programa imperativo é mantido em variáveis de programa que são associadas com localizações de memória que correspondem a um endereço e um valor de armazenamento. O valor da variável pode ser acessado direta ou indiretamente, e pode ser alterado através de um comando de atribuição. O comando de atribuição introduz uma dependência de ordem no programa: o valor de uma variável é diferente antes e depois de um comando de atribuição. Além disso, o significado (efeito) de um programa depende da ordem na qual os comandos são escritos e executados. Já as funções de linguagens de programação imperativas são descritas como algoritmos que especificam como processar um intervalo de valores, a partir de um valor de domínio, com uma série de passos prescritos. A repetição, ou laço, é usada extensivamente para processar os valores desejados. Laços são usados para varrer uma seqüência de localizações de memória tal como vetores, ou para acumular um valor em uma variável específica. Assim, devido às suas características, linguagens imperativas tem sido chamadas de "baseadas em comandos" ou "orientada a atribuições" [GHE 97].

## 2.2. Programação Imperativa

Nesta seção (2.2), através de exemplos, são apresentadas as principais características das linguagens de programação imperativas. As linguagens escolhidas para realização de uma análise comparativa são:

- Fortran,
- Pascal,
- C,
- Ada.

Em cada uma destas linguagens são abordados os seguintes tópicos:

- Valores e tipo;
- Expressões;
- Comandos e seqüências (= estruturas de controle);
- Declarações;
- Procedimentos e funções.

| | **FORTRAN** | **PASCAL** |
|---|---|---|
| - valores e tipos<br>- expressões | - tipos: integer, real, double precision, complex, logical; vetor: dimension <nome> (dim1, dim2, dim3), real, integer<br>- constantes lógicas: .true., .false.<br>- operadores: **, *, /, +, -, .ge., .gt., .le., .lt., .eq., .ne., .not., .and., .or. | - tipos: simples: boolean, integer, char, real; estruturados: array, file, record, set<br>- expressões:<br>boolean: and, or, not, xor<br>integer: +, -, *, div, mod, =, >=, <>. abs, sqr, trunc, round<br>string: var a: string; a = 'abc';<br>file: type arq = file of integer; |
| - comandos<br>e seqüências | if (<exp>) then<br>...<br>end if<br>if (<exp>)<br>...<br>else if ( ) then<br>...<br>else<br>...<br>end if<br>- comandos I/O: open, close, read, write, print, rewind, endfile<br>- goto, continue, pause, stop | - comandos simples: write, writeln, read, clrscr, gotoxy, delay, readkey, upcase<br>if-then, if-then-else<br>case <exp> of<br>case <op1>:<br>case <op2>:<br>else<br>end<br>for x := <inic> to\|downto <fim> do<br>while <cond> do<br>begin<br>...<br>end; |
| - declarações<br>- procedimentos<br>e funções | - declaração var.: <tipo> <id><br>- funções: function <id> (<parâm.>)<br>- proc.: procedure <id> (<p.>) | - declaração var.: var <id>: <tipo>;<br>- procedimentos e funções:<br>procedure <id> (<parâm.>);<br>function <id> (<parâm.>): <tipo>; |
| | versões anteriores a FORTRAN 90: somente letras maiúsculas; outras versões traduzem para maiúsculas durante a compilação | não é case sensitive |

| | C | ADA |
|---|---|---|
| - valores e tipos<br>- expressões | - tipos: char, int, float, double, struct, union<br>- operadores: -, +, *, /x, %, ++, >, >=, =, !=, &&, \|\|, !, &<br>bit a bit: &, \|, ^, ~, >>, << | - tipos:<br>array: <id> array(x..y) of <tipo><br>integer, natural, positive, character, boolean, string: type string is array(x..y) of character; float<br>- operadores: =, /=, >, >=, +, -, abs, **, and, or, xor, not |
| - comandos<br>e seqüências | if (<exp.>) <comandos><br>else <comandos><br>for (inic; cond; incremento)<br><br>switch (<exp>){<br>case <op1>: ...; break;<br>case <op2>: ...; break;<br>default: ...;<br>}<br>while (<cond.>){ ....;}<br>do<br>....<br>while (<cond.>);<br>return <exp>, goto <tótulo>, break | if <cond> then case <exp> is<br>elsif <cont> then <alt. 1><br>else <alt. 2><br>end if; end case;<br><br>when <lista escolha> => <com.><br>when <others> => <comandos><br><br>while <cond.> loop<br><comandos><br>end loop;<br>for <id> in <interv> loop<br><comandos><br>end loop |
| - declarações<br>- procedimentos<br>e funções | - declaração var.: <tipo><br><lista variáveis>;<br>- constantes: const <tipo> <id> = <valor>;<br>- funções: <tipo> <id> (<parâm>) | - declaração var.: <id>: <tipo>;<br>- procedimentos:<br>procedure <id> (<parâm>) is<br>- funções:<br>function <id> (<p.>) return <tipo> |
| | é case sensitive | |

## 2.3. Estudo de Caso: *AWK*

O material desta seção está disponível em http://www.inf.pucrs.br/~manssour/AWK/index.html