



Introdução ao Paradigma Imperativo

Prof. Tony Alexander Hild

Paradigmas de Linguagem de Programação – 3 CC – Unicentro – 2013

Definição da paradigma

pa·ra·dig·ma

(grego parádeigma, -atos)

substantivo masculino

1. Algo que serve de exemplo geral ou de modelo. = PADRÃO
2. [Gramática] Conjunto das formas que servem de modelo de derivação ou de flexão. = PADRÃO
3. [Linguística] Conjunto dos termos ou elementos que podem ocorrer na mesma posição ou contexto de uma estrutura.

"paradigma", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2013, <http://www.priberam.pt/dlpo/paradigma> [consultado em 27-09-2013].

Definição de imperativo

im·pe·ra·ti·vo

adjetivo

1. Que impera ou manda.
2. Autoritário, arrogante.

substantivo masculino

3. [Gramática] Modo do verbo que exprime ordem, exortação ou pedido.
4. Princípio que tem o caráter de obrigação imperiosa.

"imperativo", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2013, <http://www.priberam.pt/dlpo/imperativo> [consultado em 27-09-2013].

Relembrando a história...

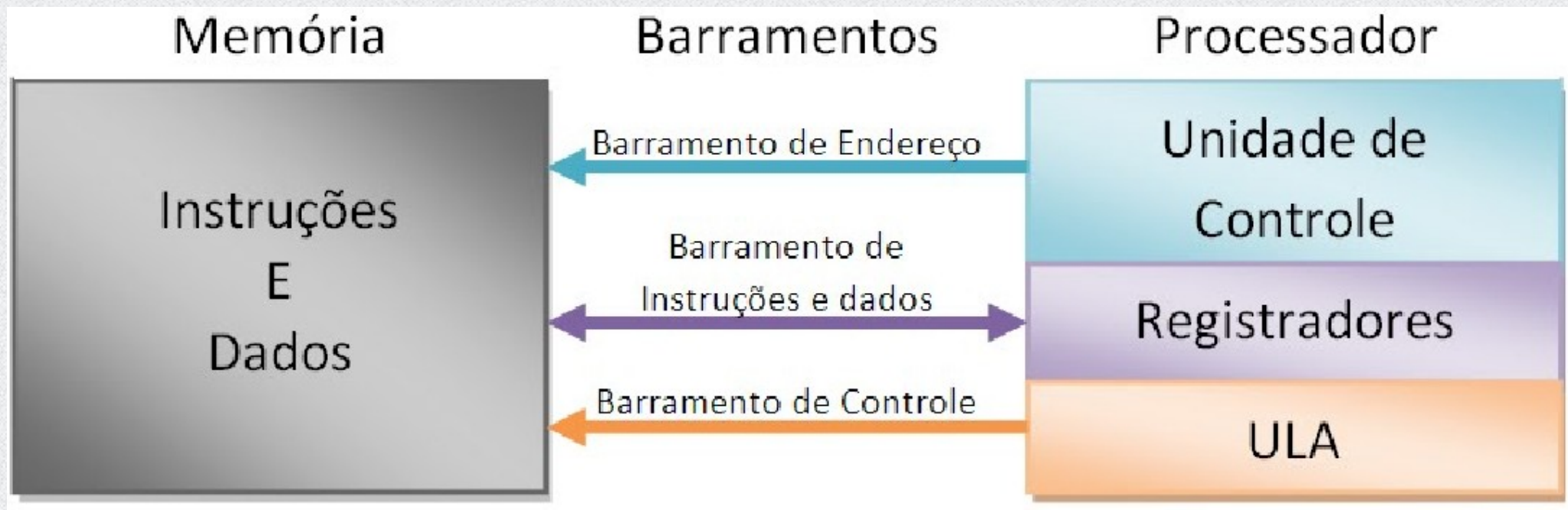
- **John von Neumann:**
 - Primeira pessoa a documentar os conceitos básicos de computadores de programas armazenados;
 - Famoso matemático húngaro que foi para os EUA em 1930 e interessou-se por computadores enquanto participava no desenvolvimento da bomba de hidrogênio.



O computador “von Neumann”

- Uma unidade de memória: é capaz de armazenar dados e instruções:
 - Acesso aleatório;
- Internamente, os dados e as instruções são armazenadas no mesmo espaço de endereço e são indistinguíveis;
- Uma unidade de cálculo (ALU);
- Uma unidade de controle (CPU);
 - Programa armazenado → conjunto de instruções;
- Dualidade entre instruções e dados → programas podem automodificar-se;
- von Neumann delineou essa estrutura em um documento conhecido como o “First Draft of a Report on the EDVAC” de junho de 1945.

O computador “von Neumann”



Completude de Turing

- Alan Turing: um matemático britânico e um dos primeiros cientistas da computação;
- Uma linguagem é Turing completa se puder ser utilizada para implementar qualquer algoritmo;
- Muito importante para o estudo da computabilidade.



O que torna uma linguagem imperativa?

- Programas escritos em linguagens de programação imperativas consistem em:
 - Um estado do programa;
 - Instruções que alteram o estado do programa.
- Instruções do programa são "imperativas", no sentido gramatical dos verbos imperativos que expressam um comando.

Primeiro faça isto, depois faça aquilo.


História das Linguagens Imperativas

- Primeiras linguagens imperativas: linguagens de montagem (assembly);
- 1954-1955: Fortran (**FOR**mula **TRAN**slator):
 - John Backus desenvolvido para IBM 704.
- Final de 1950: Algol (**ALGO**rithmic **L**anguage);
- 1958: Cobol (**C**ommon **B**usiness **O**riented **L**anguage), desenvolvido por uma comissão do governo, Grace Hopper muito influente.

Elementos das linguagens imperativas

- Definições de tipo de dados;
- Declarações de variáveis:
 - São normalmente "tipadas" estaticamente ou dinamicamente:
 - Tipos básicos de dados (e.g., int, float, boolean, char);
 - Tipos de dados compostos (structs, arrays).
- Expressões e declarações de atribuição;
- Declarações de controle de fluxo (geralmente estruturados);
 - Desvios condicionais, laços;
 - As declarações são comandos e sua ordem é fundamental para a execução correta;
- Escopos lexicais e blocos:
 - Objetivo: proporcionar localidade de referência.
- Declarações e definições de procedimentos e funções (isto é, blocos parametrizados);
- Manipulação de erros e exceções;
- Comandos de E/S;
- Estruturas de dados.

Características das linguagens imperativas

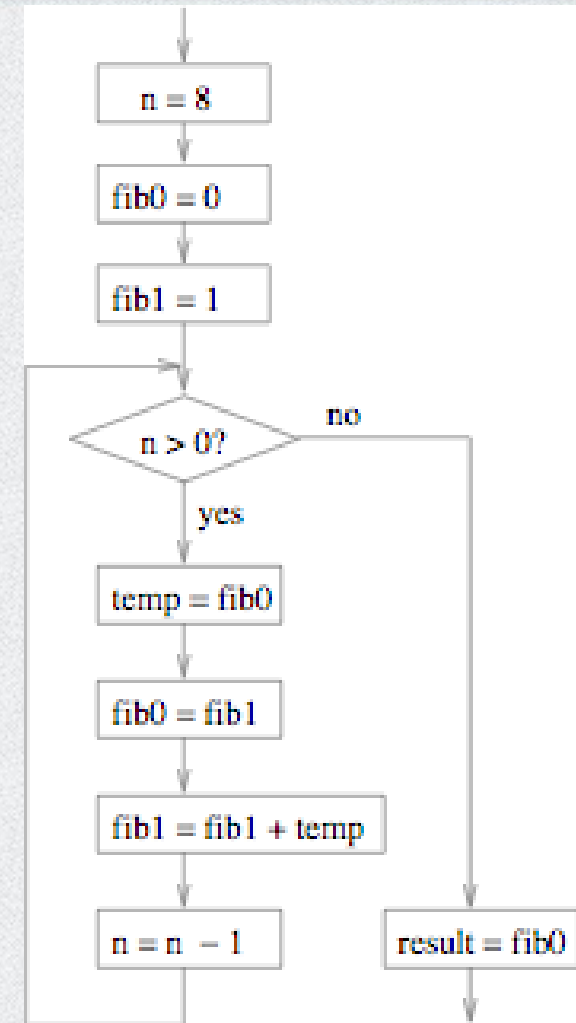
- São Turing completas se suportarem inteiros, operadores aritméticos básicos, atribuições, sequenciamento, laços e desvios condicionais;
- O estado do programa é mantido em variáveis associadas à localizações de memória:
 - Endereço e um valor de armazenamento;
- O valor da variável pode ser acessado direta e indiretamente, e pode ser alterado através de um comando de atribuição;
 - `a = 1;`
 - `&a = 1;`
- Uma atribuição introduz uma dependência de ordem no programa:
 - o valor de uma variável é diferente antes e depois de um comando de atribuição;
 - `print(a) => 1;`
 - `a = 2;`  dependência de ordem
 - `print(a) => 2;`

Características das linguagens imperativas

- Repetição ou laço ou loop:
 - Usada extensivamente para processar valores ou acumular um valor em uma variável específica; ou
 - Usada para varrer uma sequência de localizações de memória tal como vetores.
- Funções:
 - Algoritmos que especificam como processar um intervalo de valores, a partir de um valor de domínio, com uma série de passos prescritos;
- Comandos:
 - Semelhantes às instruções de máquina nativas de um hardware de computador tradicional
 - o modelo de von Neumann-Eckley;
- O paradigma imperativo é predominante nas LPs, pois tais linguagens são mais fáceis de traduzir para uma forma adequada para execução na máquina;
- Usa-se abstração procedural e refinamento gradual para melhorar um programa imperativo.

Gráfico de fluxo

- Usado para modelar programas imperativos;
- Com base nas três declarações de controle que são essenciais para se obter toda a capacidade de uma máquina de Turing;
- Precursor da UML e outras técnicas modernas;
- Criado para descrever fluxos de processos em geral.



Abstração Procedural

- Nicholas Wirth descreveu programas (imperativos) como "algoritmos + estruturas de dados";
- Algoritmos tornam-se programas através do processo de abstração procedural e refinamento gradual;
- Bibliotecas de funções reutilizáveis facilitam este processo (funções = procedimentos);
- Programação imperativa + procedimentos = programação procedural.

Abstração Procedural

- A abstração procedural permite que o programador se preocupe principalmente com as interfaces entre as funções (procedimentos) e o que elas calculam, ignorando os detalhes de como o cálculo é realizado;
- Abstração permite pensar sobre o que está sendo feito, e não como é implementado.

Refinamento gradual (passo a passo)

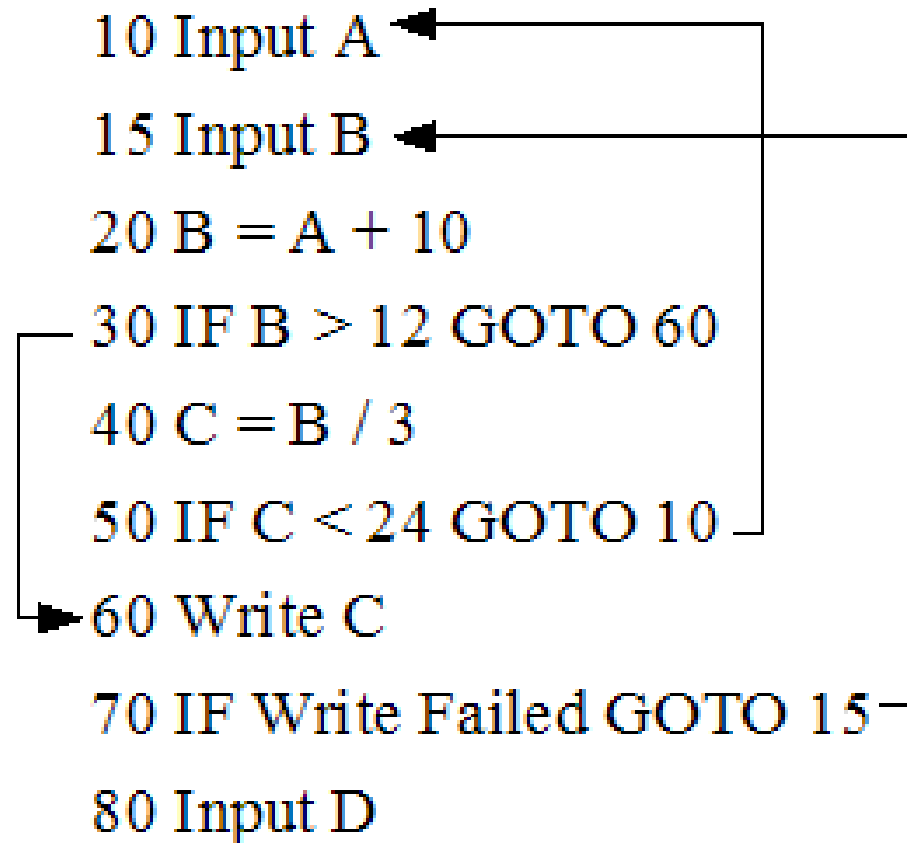
- Refinamento gradual (também chamado de decomposição funcional) utiliza abstração procedural para desenvolver um algoritmo a partir de sua forma mais geral, (a abstração) para sua implementação específica (método top-down);
- Os programadores começam com a descrição do que o programa deve fazer, incluindo E/S, e repetidamente quebram o problema em partes menores, até que os subproblemas possam ser expressados em termos dos estados primitivos e dos tipos de dados da linguagem.

Programação Estruturada

- Uma abordagem disciplinada para projetos de programas imperativos;
- Usa abstração procedural e projeto *top-down* para identificar os componentes do programa (também chamados de módulos ou estruturas);
- Estruturas do programa combinadas em um número limitado de formas, de modo que a compreensão de como funciona cada estrutura significa entender como funciona o programa;
- O fluxo de controle do programa é baseado em decisões, sequências, laços, mas...
 - Não usa instruções goto.
- Os módulos são desenvolvidos, testados separadamente e depois integradas no conjunto do programa.

Instruções goto

Código Spaghetti



Atribuições

- Lembre-se: linguagens imperativas operam alterando o estado do programa. Isso é feito usando comandos de atribuição destrutivos.
- Formato geral:
 - destino = expressão
- Operadores de atribuição: = ou :=
- Baseado em operações de máquina, tais como MOV ou STO.

Semântica de Atribuição

- Avalia a expressão para obter um único valor;
- Copia o valor da expressão para o destino;
- Programas imperativos puros implementam a semântica de cópia (em oposição à semântica de referência utilizadas em linguagens orientadas a objetos).

Expressões

- Expressões representam um valor e têm um tipo;
- Compreender as expressões significa entender a precedência do operador, sobrecarga de operadores, *casting* e conversão de tipo, entre outras questões;
- Expressões aritméticas simples são baseados em operadores aritméticos (DIV, MUL, etc) de linguagem de máquina (LM).

Expressões

- Os operadores lógicos são baseadas em instruções de LM semelhantes (AND, XOR, ...);
- LM suporta tipos de dados indiretamente através de diferentes instruções:
 - add/sub/mul/div de inteiros
 - VS
 - add/sub/mul/div de ponto flutuante, por exemplo.
- Internamente, não há maneira visível para distinguir inteiros de ponto flutuantes de caracteres;
- Línguas a partir do Fortran fornecem alguma proteção contra erros de tipo, e alguma capacidade para definir novos tipos.

Linguagens Imperativas X Declarativas

- Linguagens **imperativas** (Java, Pascal, C/C++, C#):
 - Especificam uma sequência de operações para o computador executar.
- Linguagens **declarativas** (SQL, Haskell, Prolog)
 - Descrevem o espaço de soluções;
 - Fornecem o conhecimento necessário para chegar lá;
 - Não descrevem os passos necessários para chegar lá.
 - Linguagens funcionais e linguagens lógicas são declarativas.

Vantagens e Desvantagens

- **Vantagens:**

- Eficiência (embute o modelo de Von Neumann);
- Mais fácil de traduzir para a linguagem de máquina;
- Paradigma dominante e bem estabelecido;
- Modelagem “Natural” de aplicações do mundo real;
- Também é muito flexível.

- **Desvantagens:**

- Descrições demasiadamente operacionais;
- Focaliza o “como” e não o “quê”;
- Relacionamento indireto com a E/S (indução a erros/estados).

Exemplos

- Ada
- Algol
- C
- Pascal

ADA

```
type Day_type is range 1 .. 31;
type Month_type is range 1 .. 12;
type Year_type is range 1800 .. 2100;
type Hours is mod 24;
type Weekday is (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

type Date is
  record
    Day : Day_type;
    Month : Month_type;
    Year : Year_type;
  end record;
```


ADA

```
-- while a is not equal to b, loop.
while a /= b loop
  Ada.Text_IO.Put_Line ("Waiting");
end loop;

if a > b then
  Ada.Text_IO.Put_Line ("Condition met");
else
  Ada.Text_IO.Put_Line ("Condition not met");
end if;

for i in 1 .. 10 loop
  Ada.Text_IO.Put ("Iteration: ");
  Ada.Text_IO.Put (i);
  Ada.Text_IO.Put_Line;
end loop;

loop
  a := a + 1;
  exit when a = 10;
end loop;

case i is
  when 0 => Ada.Text_IO.Put ("zero");
  when 1 => Ada.Text_IO.Put ("one");
  when 2 => Ada.Text_IO.Put ("two");
  -- case statements have to cover all possible cases:
  when others => Ada.Text_IO.Put ("none of the above");
end case;

for aWeekday in Weekday'Range loop
  Put_Line ( Weekday'Image(aWeekday) );
  if aWeekday in Working_Day then
    Put_Line ( " to work for " &
              Working_Hours'Image (Work_Load(aWeekday)) ); -- access into a lookup table
  end if;
end loop;
```

ALGOL

```
proc abs max = ([,]real a, ref real y, ref int i, k)real:
comment The absolute greatest element of the matrix a, of size [a by 2[a
is transferred to y, and the subscripts of this element to i and k; comment
begin
  real y := 0; i := [a; k := 2[a;
  for p from [a to [a do
    for q from 2[a to 2[a do
      if abs a[p, q] > y then
        y := abs a[p, q];
        i := p; k := q
      fi
    od
  od;
  y
end # abs max #
```


C

```
long some_function();  
/* int */ other_function();  
  
/* int */ calling_function()  
{  
    long test1;  
    register /* int */ test2;  
  
    test1 = some_function();  
    if (test1 > 0)  
        test2 = 0;  
    else  
        test2 = other_function();  
    return test2;  
}
```

Pascal

```
while a <> b do WriteLn('Waiting');  
  
if a > b then WriteLn('Condition met') {no semicolon allowed!}  
    else WriteLn('Condition not met');  
  
for i := 1 to 10 do {no semicolon for single statements allowed!}  
    WriteLn('Iteration: ', i);  
  
repeat  
    a := a + 1  
until a = 10;  
  
case i of  
    0 : Write('zero');  
    1 : Write('one');  
    2 : Write('two');  
    3,4,5,6,7,8,9,10: Write('?')  
end;
```

```
type  
    byte           = 0..255;  
    signed_byte    = -128..127;  
    string          = packed array[1..255] of char;
```