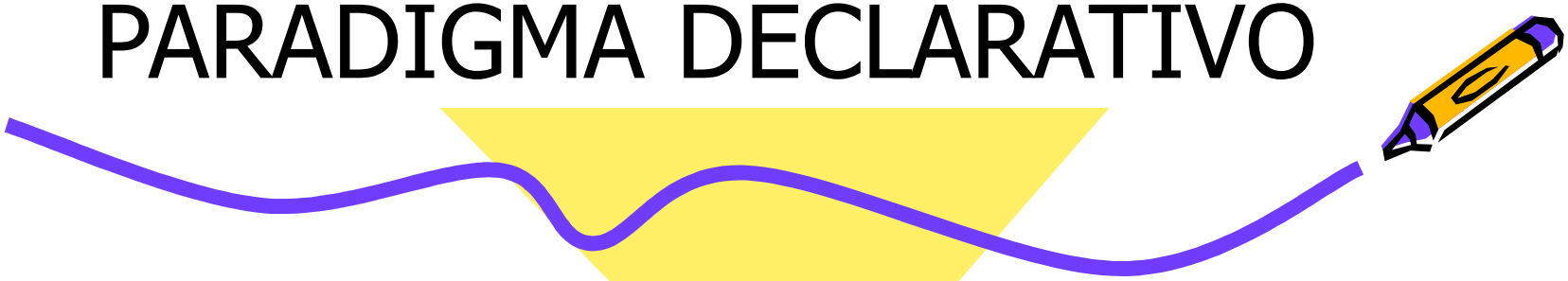
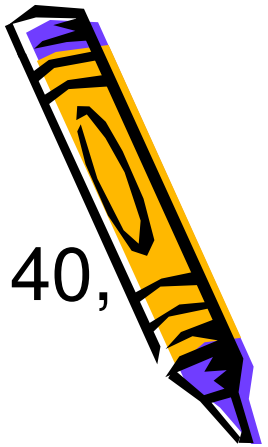
Paradigmas de Programação

PARADIGMA IMPERATIVO
X
PARADIGMA DECLARATIVO



Paradigma Imperativo



- Mais antigo e bem desenvolvido, década de 40, e de certa forma, ainda dominante
- Influência da arquitetura de computadores preponderante de **Von Neumann**
- Atribuição

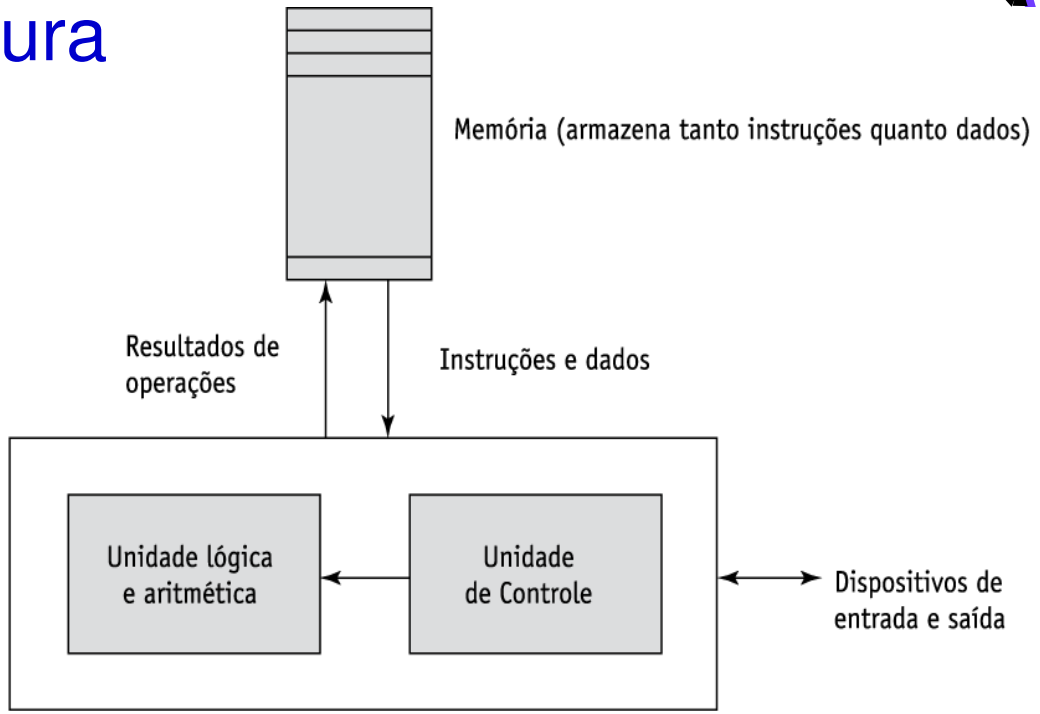
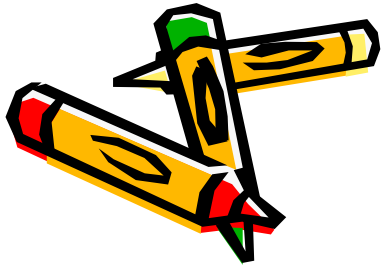


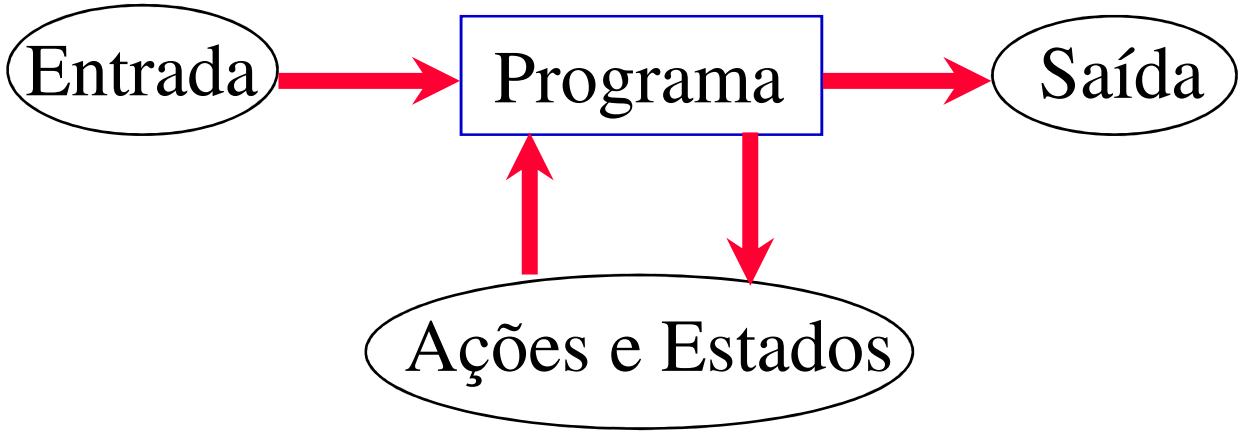
Figura 1.1 A arquitetura de computadores de von Neumann.

(SEBESTA, 2003)

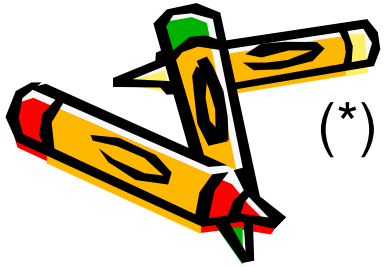


Paradigma Imperativo*

- Conceito baseado em **estados** (modelado por variáveis) e **ações** manipuladoras destes estados

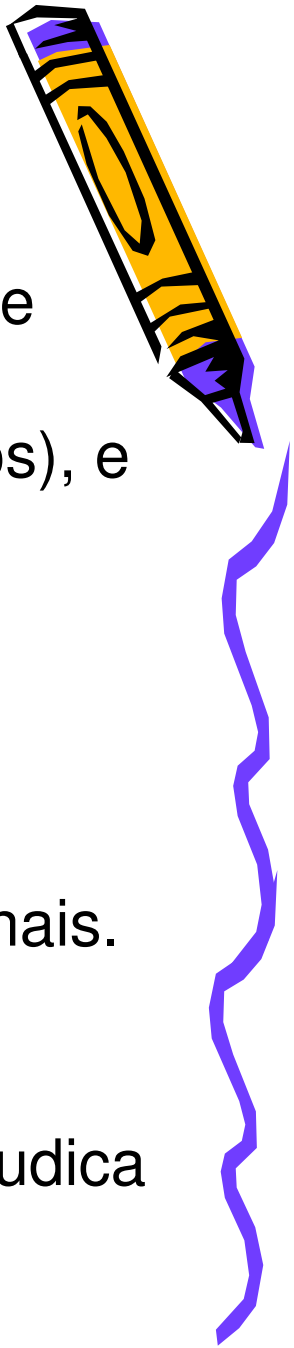


- Informa **como** as instruções devem ser executadas
- Sequência de ordens: "*Primeiro faça isso, depois faça aquilo*"



(*) *Imperare*, latim = comandar

Linguagens Imperativas



- Conceitos principais:
  - variáveis (representam simbolicamente células de memória),
  - declarações de **atribuição** (transmissão de dados), e
  - forma iterativa de declarações de repetição.
- Comandos:
  - de **atribuição**,
  - condicionais, e
  - de ramificação (*goto**), condicionais e incondicionais.

=> COMPLETA QUANTO A TURING

(*) Dijkstra, 68: uso excessivo de goto prejudica confiabilidade

Linguagens Imperativas

- Ser “Completa quanto a Turing”[1] significa:
  fornecer base efetiva de recursos para implementação de qualquer algoritmo que possa ser projetado
- E deve conter:
  - Estruturas de controle
  - Entrada/saída
  - Manipulação de exceções e erros
  - Abstração procedural
  - Expressões e atribuição
  - Suporte de biblioteca para estruturas de dados

(1) Linguagens de outros paradigmas tb podem ser, no sentido de que qualquer uma é igualmente capaz de expressar qualquer algoritmo.

Linguagens Imperativas

Programas = algoritmos + estruturas de dados (Wirth, 76)

Abstração procedural:

> "O processo de abstração procedural permite ao programador se preocupar principalmente com a interface entre a função e o que ela calcula, ignorando os detalhes de como o cálculo é executado."

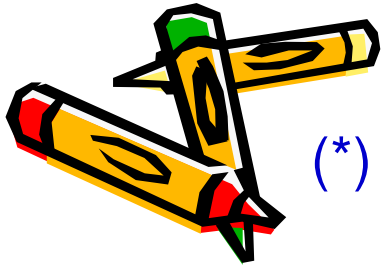
Refinamento gradual* (Wirth, 73):

> "processo que utiliza a abstração procedural desenvolvendo um algoritmo da sua forma mais geral para uma implementação mais específica."

Ex.: sort (list, len);

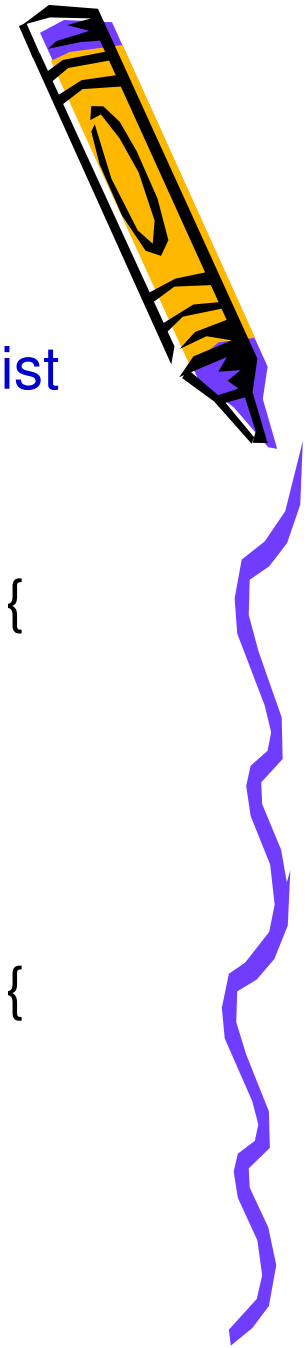
// especifica E/S mas não como realiza ação

// código contém qq algoritmo de ordenação



(*) Ou refinamento em passos, decomposição funcional

Linguagens Imperativas

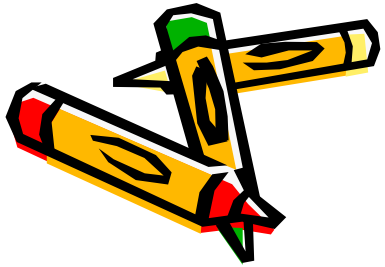


Implementações do algoritmo sort:

A) foreach i in the sequence of indices of list {
    list[i] = minimum elemento in remaining list
}

B) foreach i in the sequence of indices of list {
    foreach j > i in the sequence of indices of list {
        list[i], list[j] = min, max of list[i], list[j]
    }
}

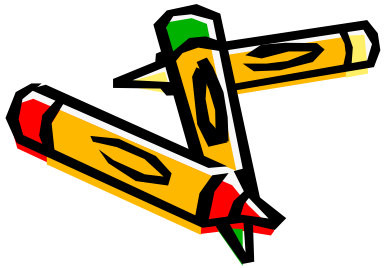
C) foreach i in the sequence of indices of list {
    foreach j > i in the sequence of indices of list {
        if list[i] < list[j] {
            swap list[i], list[j]
        }
    }
}



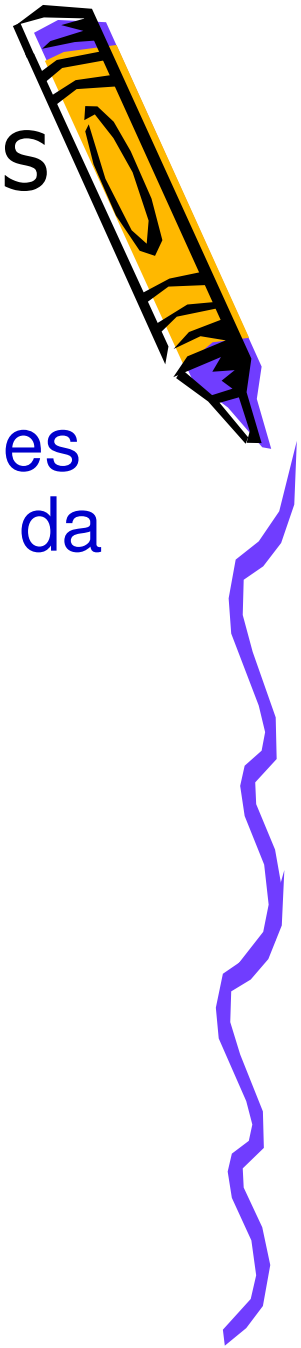
Linguagens Imperativas

Algoritmo refinado pode então ser codificado razoalvelmente de forma direta – em C poderia ser:

```
void sort (Type list, int len) {
    for (int i=0; i<len; i++)
        for (int j=i+1; j<len; j++)
            if ( list[ i ] < list[ j ] ) {
                Type t = list[ j ];
                list[ j ] = list[ i ];
                list[ i ] = t;
            }
}
```

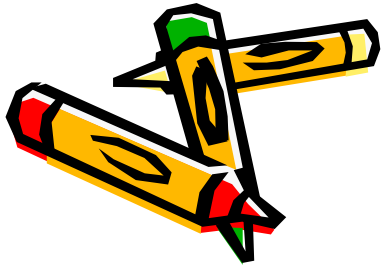


Ling. Imperativas Não Estruturadas

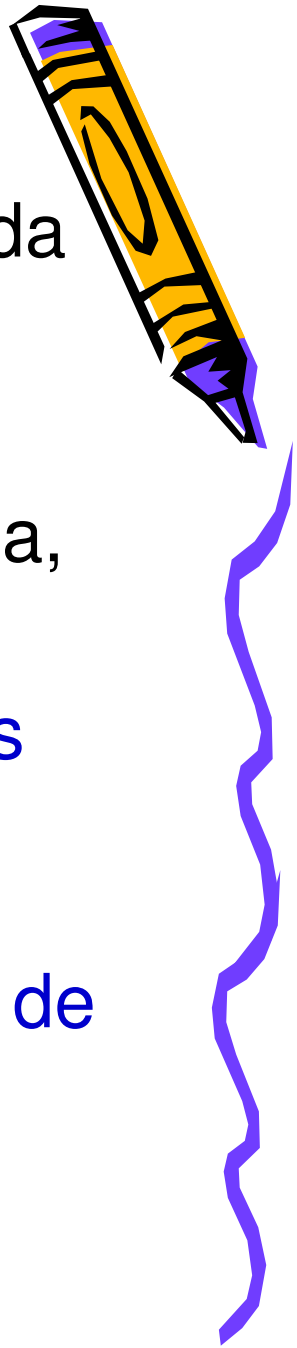


- Assembly, FORTRAN e BASIC
- Versões originais FORTRAN => *goto* para repetição e seleção de execução de instruções => dificuldade na leitura e acompanhamento da execução.
- Exemplo baseado na sintaxe do Pascal:

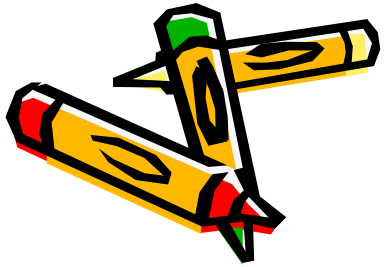
```
        read(x);
2:      if x = 0 then goto 8;
        writeln(x);
4:      read(next);
        if next = x then goto 4;
        x := next;
        goto 2;
8: ...;
```



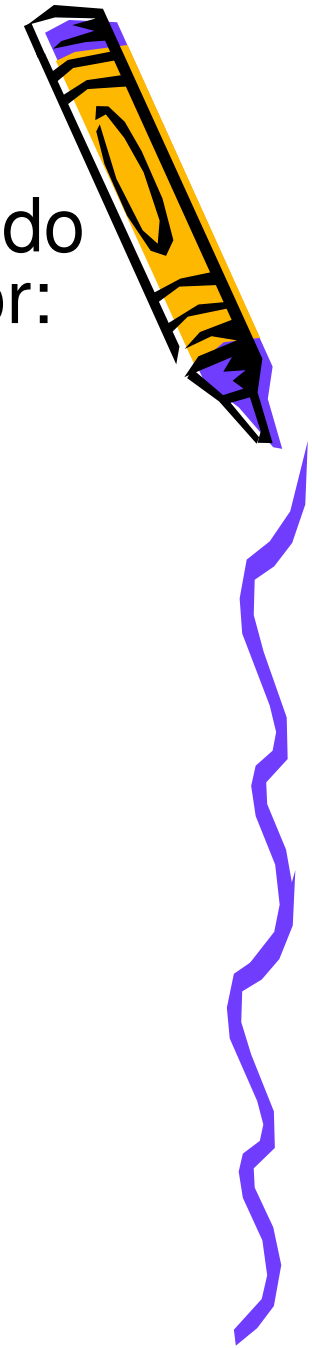
Ling. Imperativas Estruturadas



- Objetivo: facilitar leitura e acompanhamento da execução de algoritmos.
- Não usam *goto* !!!!
- Instruções em blocos = unidades de programa, abstraindo-se das suas estruturas internas.
- Blocos de instruções podem ser selecionados para execução através de declarações de seleção como *if...else*, ou repetidamente executados através de declarações de repetição como *while*.

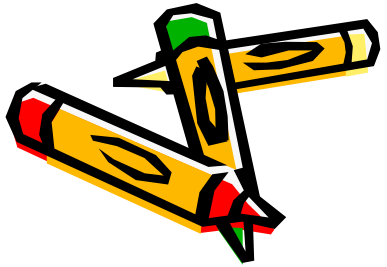


Ling. Imperativas Estruturadas

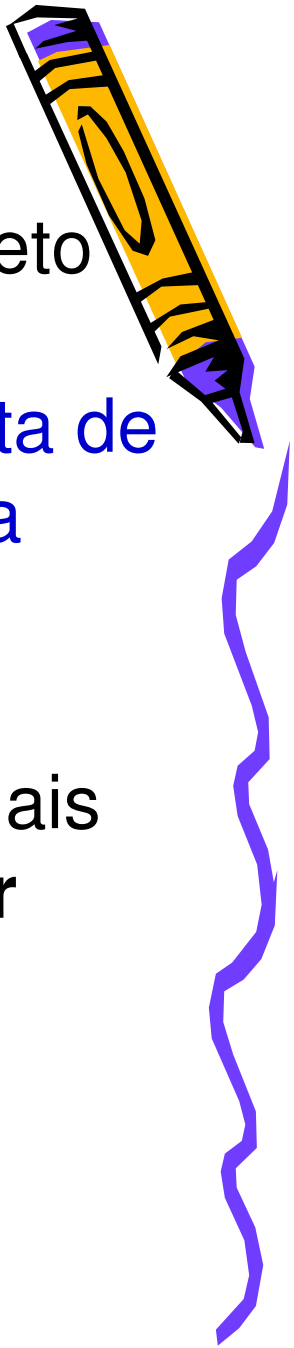


- Exemplo de código estruturado, na sintaxe do Pascal, correspondente ao exemplo anterior:

```
read(x);
while x <> 0 do begin
        writeln(x);
        repeat
                read(next);
        until next <> x;
        x := next;
end;
```

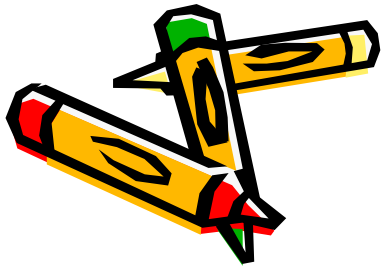


Vantagens X Desvantagens



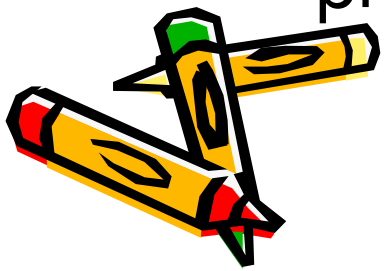
- Eficiência (embute modelo de V*on Neumann*)
- Mais fácil de traduzir para a linguagem de máquina
- Paradigma dominante e bem estabelecido

- Relacionamento indireto entre E/S (indução a erros/ estados) => falta de legibilidade semântica (mix comportamento, interface e dados)
- Descrições operacionais focalizam **como fazer**



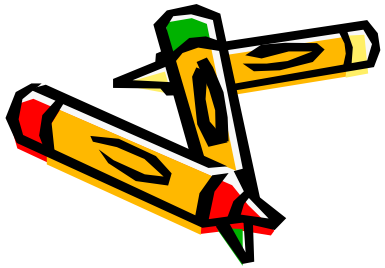
Programação Procedural

- Sinônimo de Programação Imperativa
- Deve estar baseada e suportar o conceito de *procedimentos* (rotinas) e possuir sintaxe para defini-los.
- A maioria das linguagens procedurais também são linguagens imperativas, pois fazem referências explícitas ao estado do ambiente de execução.
    - Geralmente é melhor escolha do que programação sequencial e não estruturada (complexidade média e facilidade de manutenção).



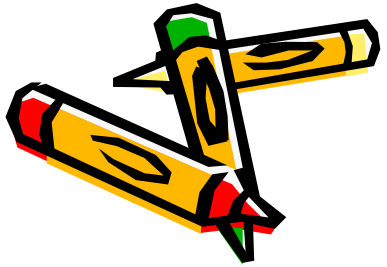
Programação procedural

- Possíveis benefícios são:
  - Habilidade de reutilizar o mesmo código em diferentes lugares no programa sem copiá-lo
  - Forma mais fácil de organizar o fluxo do programa que uma coleção de comandos *goto* ou *jump* (programa grande e complicado => código *spaguetti*)
  - A habilidade de ser fortemente modular e estruturado



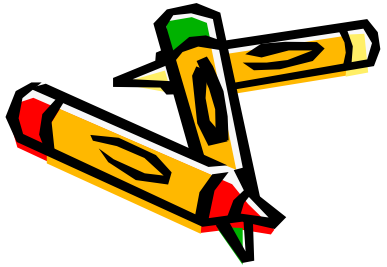
Programação Orientada a Objetos

- Abstração de dados (ou TDAs), Simula67
- Estende a noção de tipo (encapsulamento)
- Objetivo: empacotar os tipos de dados e suas funções juntos em único módulo, com as funções proporcionando uma interface pública.
- Módulos permitem separação da interface lógica (especificação) da implementação => ocultação de informação



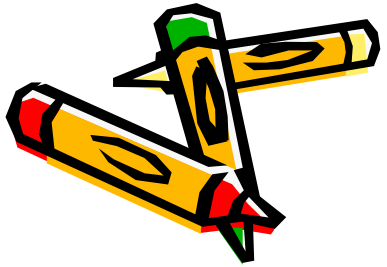
Programação Orientada a Objetos

- Problemas dos 80s: inicialização e finalização automática de valores, módulos não proporcionavam maneira simples de estender abstração de dados, necessidade de atender aplicações embarcadas e GUIs,...
- POO surgiu como estilo popular de programação no qual a decomposição de objetos era o foco, no lugar da decomposição funcional e da abstração de dados



Programação Orientada a Objetos

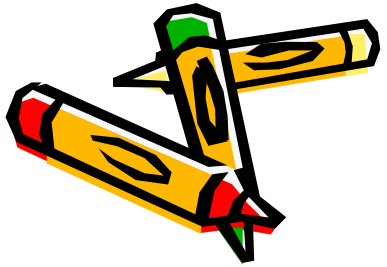
- Classes
- Visibilidade e ocultamento de informação
- Herança simples e múltipla
- Polimorfismo
- Modelos (templates)
- Classes abstratas
- Interfaces
- Reflexão



Vantagens X Desvantagens

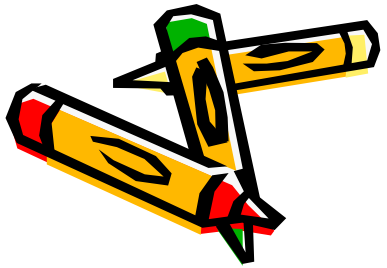
- As mesmas do imperativo
- Classes estimulam modularidade, reusabilidade, extensibilidade
- Mercado crescente

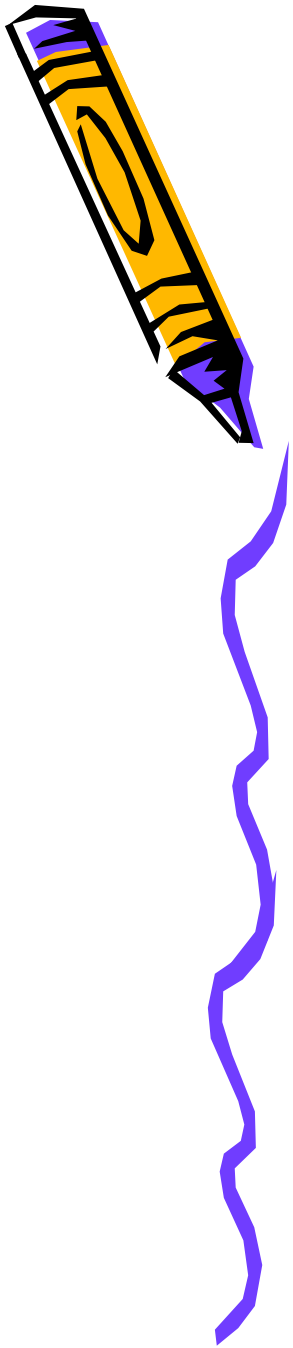
- Semelhantes as do imperativo, porém amenizadas pelas facilidade de estruturação



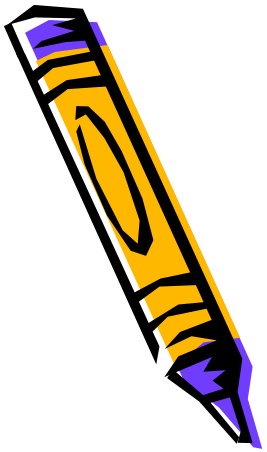
Alguns exemplos de linguagens do modelo imperativo

- Ada
- Algol
- Basic
- C
- Cobol
- Fortran
- Pascal
- **Python**
- **Java**
- **Smalltalk**
- **C++**
- Lua
- Mathematica



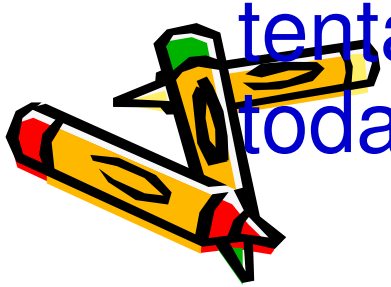


Paradigma Declarativo



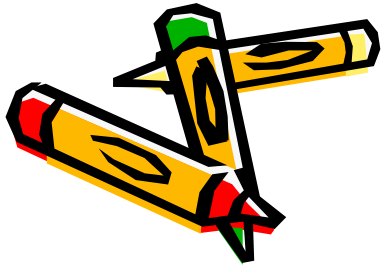
- Requer a descrição do problema:

  descrever propriedades da solução desejada – não especificar como o algoritmo deve agir
- O programa é estruturado como uma coleção de propriedades para encontrar o resultado esperado
- Exemplo: Dado um banco de dados ou um conjunto de regras, o computador tenta encontrar a solução ao casar todas as propriedades desejadas.



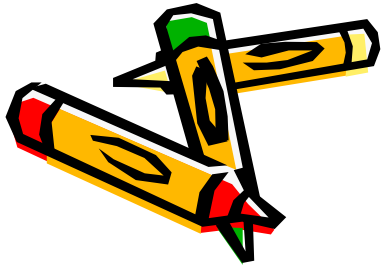
Paradigma Declarativo

- Separa lógica (**o que**) de controle (**como**)
- Estratégia: descobrir e implementar um **algoritmo geral** para solucionar o problema
- **Tarefa do programador**: desenvolver uma descrição precisa do problema, ao invés de descobrir um algoritmo para resolvê-lo
- Base de programação funcional, lógica ou restritiva

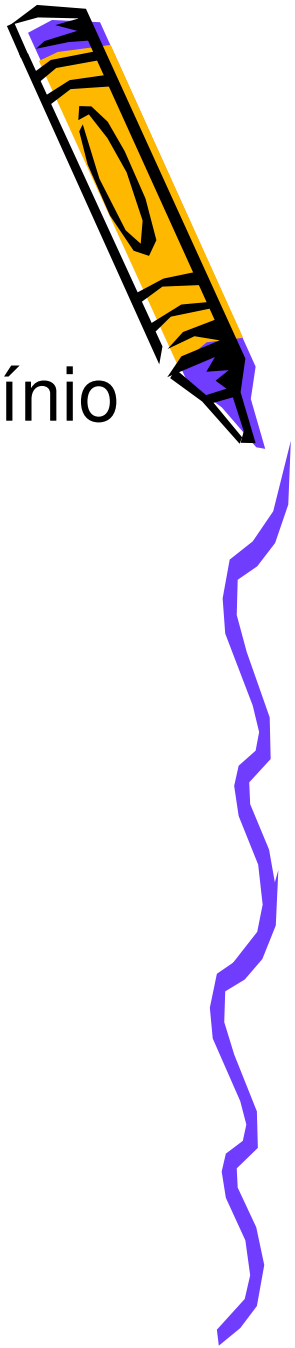


Paradigma Declarativo

- Ganhou impulso ao se descobrir que a lógica formal propicia a elaboração de um algoritmo simples para resolução de problemas, adequado ao uso em sistemas declarativos de propósito geral (programação lógica)
- Ex. dedução lógica + prolog



Alguns exemplos de linguagens do modelo declarativo

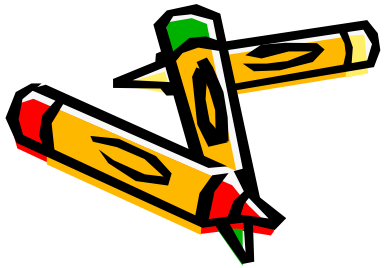


Forma popular: linguagens específicas de domínio

- XSTL (transformar documentos XML)
- SQL

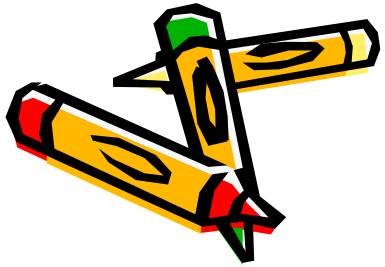
Outras linguagens:

- Erlang
- Lisp
- Haskell
- Prolog
- Mercury

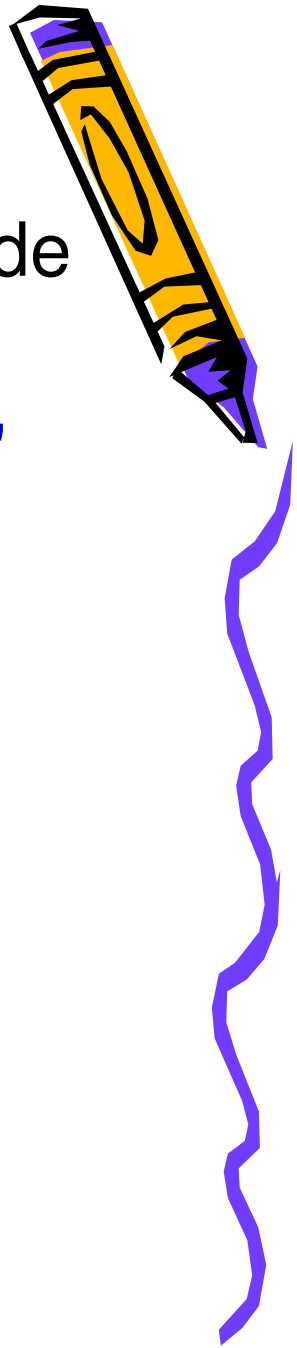


Vantagens X Desvantagens

- Tratamento e armazenamento de dados
- Programador: definições e/ou equações especificando relações
- Certa ilegibilidade de código (ex. código HTML)



Haskell



- O difundido caso do Programa Olá Mundo pode ser exemplificado da seguinte forma:

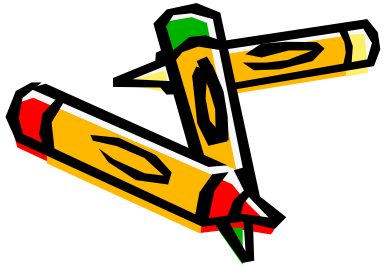
```
olamundo :: IO() olamundo = putStr "Ola mundo"
```

- A clássica definição da função fatorial:

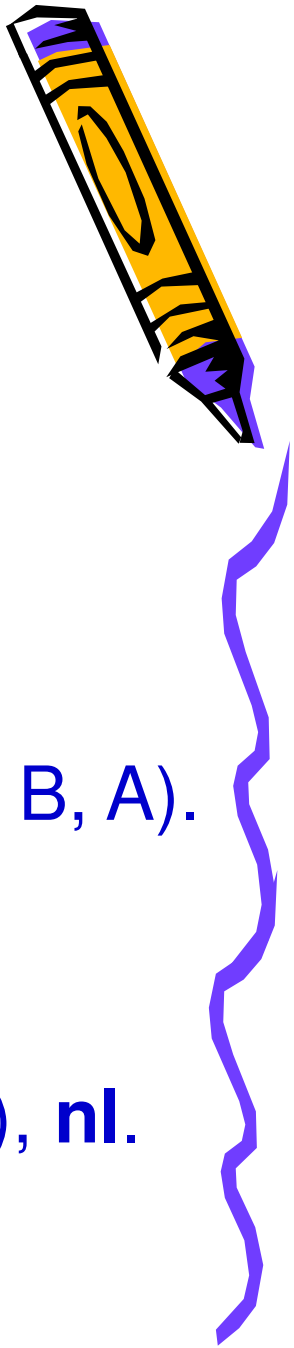
```
fatorial :: Integer -> Integer
fatorial 0 = 1
fatorial n | n > 0 = n * fatorial (n-1)
```

- Ou em uma linha:

```
fatorial n = if n > 0 then n * fatorial (n-1) else 1
```

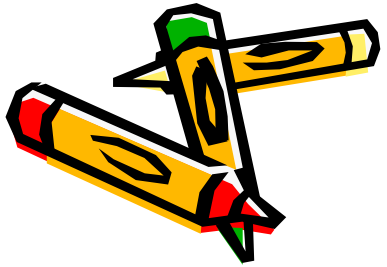


Prolog



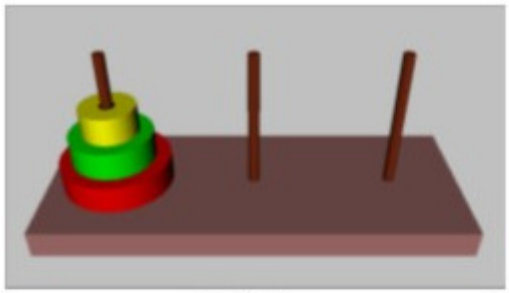
- **Torres de Hanoi**

```
hanoi(N) :- move(N, left, centre, right).
move(0, _, _, _) :- !.
move(N, A, B, C) :-
   M is N-1,
   move(M, A, C, B), inform(A, B), move(M, C, B, A).
inform(X, Y) :-
    write('move a disc from the '),write(X),
    write(' pole to the '), write(Y), write(' pole'), nl.
```

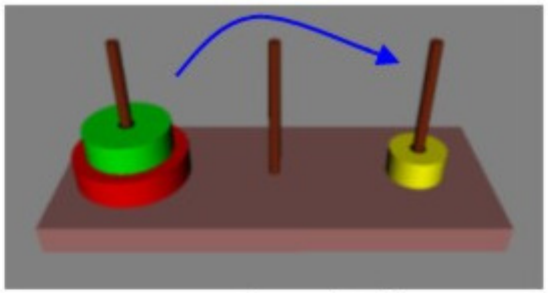


Prolog

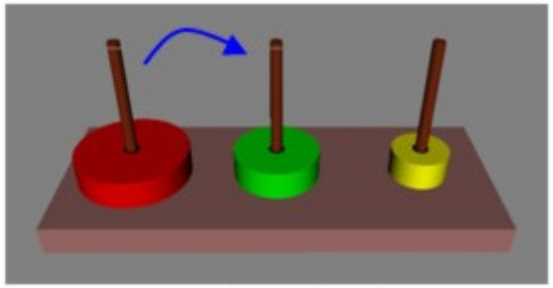
- **Torres de Hanoi**



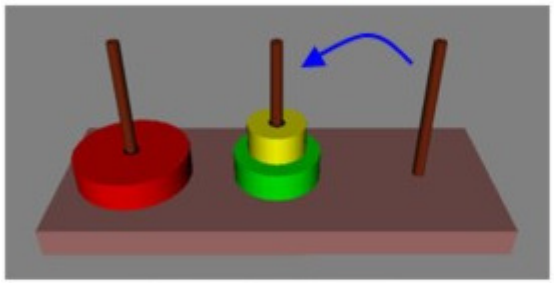
início



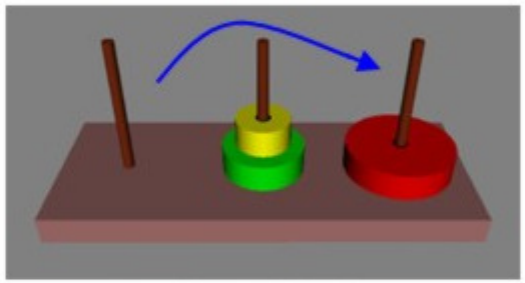
movimento 1



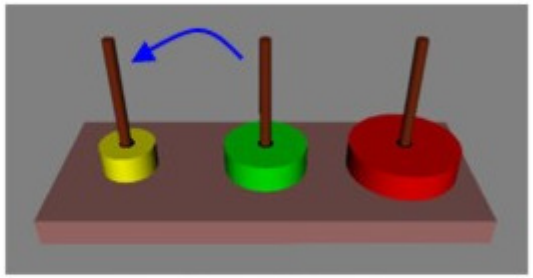
movimento 2



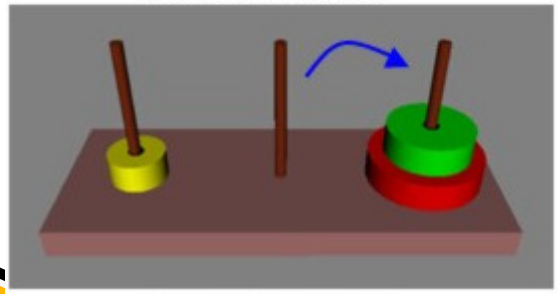
movimento 3



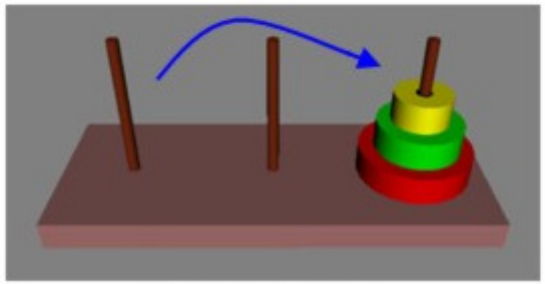
movimento 4



movimento 5



movimento 6



movimento 7