1.6 Especificação de Algoritmos

- Seqüenciamento / Paralelismo
- Ramificação / Seleção
- Iteração
- Recursão
- Modularidade

Sequenciamento / Paralelismo

Algoritmos são normalmente apresentados de forma seqüencial

>> reveja os passos seqüenciais do algoritmo de Euclides

No entanto, em muitas ocasiões, há operações que podem ser executadas de forma paralela se houver recursos para isso.

>> nestes casos a ferramenta (linguagem) que descreve o algoritmo pode explicitar isto.

Exemplo

```
<project>
    <include  file="a.xml"/>
    <parallel>
    <gridExecute host="(host)" provider="gt2"
            executable="/home/gregor/mm5"
            arguments="climate-1.dat"
            annotation="A"/>
    <gridExecute host="(host)" provider="gt3"
            executable="/home/gregor/mm5"
            arguments="climate-2.dat"
            annotation="B"/>
    <sequential>
    <gridExecute host="(host)" provider="gt2"
            executable="/home/gregor/mm5"
            arguments="climate-3.dat"
            annotation="C"/>
    <gridExecute host="(host)" provider="gt4"
            executable="/home/gregor/mm5"
            arguments="climate-4.dat"
            annotation="D"/>
    </sequential>
    </parallel>
</project>
```

Ramificação / Seleção

Este é um construtor importante na definição de algoritmos, permitindo a opção entre alternativas em função de circunstâncias durante a execução do algoritmo.

IF condição
  THEN passo1;
  ELSE passo2;

CASE variável
  variável = estado1: **THEN** passo1;
  variável = estado2: **THEN** passo2;
END_CASE

Exemplos (linguagem C)

```
    .....
main ( )
{
    int magic=123;
    int guess;
    scanf("%d", &guess); /*entrada*/
    if (guess==magic) printf
    ("*OK*");
    else printf ("*FALSO*");
}
```

```
    .....
menu ( )
{
    char ch;
    printf ("1. verifique palavra \n");
    printf ("2. corrigir  palavra \n");
    printf ("3. outro número - pular \n");
    printf ("Faça a sua escolha  \n");
    ch = getchar( ); /*entrada*/
    switch (ch) {
    case "1" : checkspelling ( );
    break;
    case "2" : correct_errors ( );
    break;
    case "3" : error ( );
    break;
    default: printf ("nenhuma opção");
                }
}
```

Iteração

Construtor que define a repetição de partes de um algoritmo:

REPEAT

passo 1;

passo 2;

....

UNTIL condição

Exemplo de iteração – linguagem C

```
main ( )
{
  int x;
  for  (x=1; x<=100; x++) printf ("%d", x);
}
```

⟷

for(inicialização; condição; incremento) comando;

ou

```
....
do {
        scanf ("%d", &num); /*lê dados via teclado*/
        } while (num>100);   /*enquanto > 100*/
```

⟷

```
do{
    comando;
  } while (condição);
```

Recursão

Construtor que também define a repetição de partes de um algoritmo.

Exemplo clássico: cálculo do fatorial de um número

```
FATORIAL (n)
IF  n=0  THEN  FATORIAL = 1
         ELSE  FATORIAL =  n . FATORIAL (n-1)
FIM
```

Exemplo de recursão – linguagem C

```
printd (n)  /* imprime decimal n */
int n;
{
int i;
if (n<0) {
    putchar ("-");
    n = - n;
        }
If ((i=n/10) != 0)
    printd (i);
putchar (n%10 + "0");
}
```

Execução com printd (123)

Inicia execução 1 com n=123 > printd(12)

Inicia execução 2 com n=12 > printd(1)

Inicia execução 3 com n=1

> i = 1/10=0

> putchar(1%10+"0") imprime 1

Volta à execução 2

> putchar(12%10+"0") imprime 2

Volta à execução 1

> putchar(123%10+"0") imprime 3

Modularidade

Um construtor que permite subdividir os algoritmos em “partes” denominadas **módulos.**

Exemplos:

procedure

function

routine

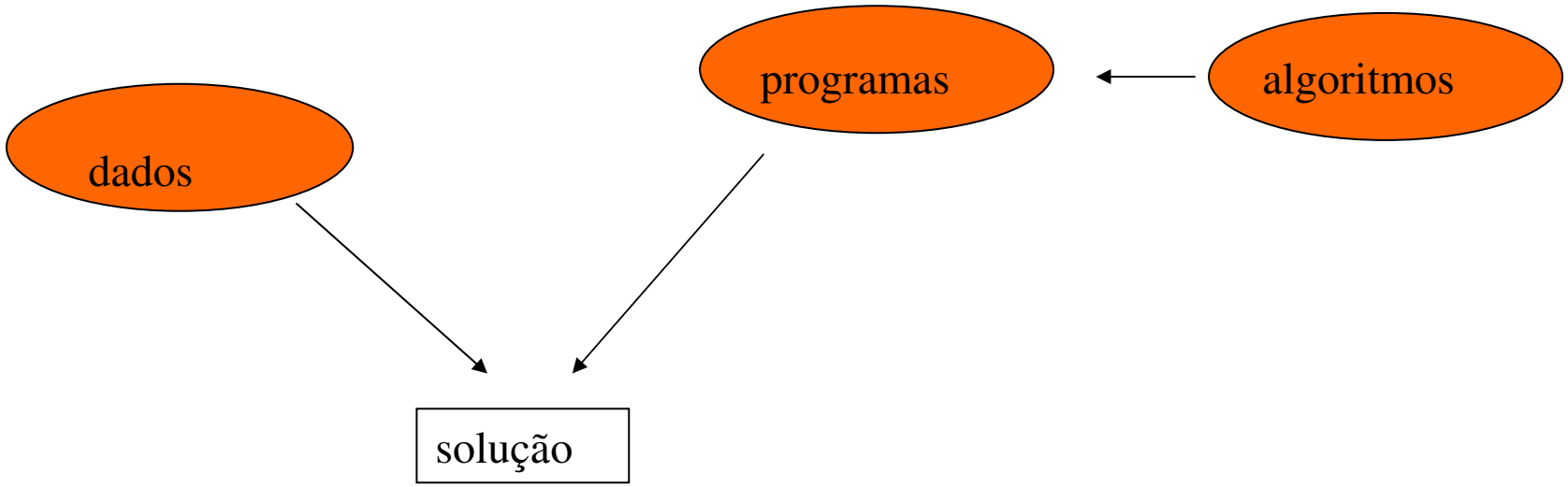
main ( )

Exemplo modularidade: linguagem C

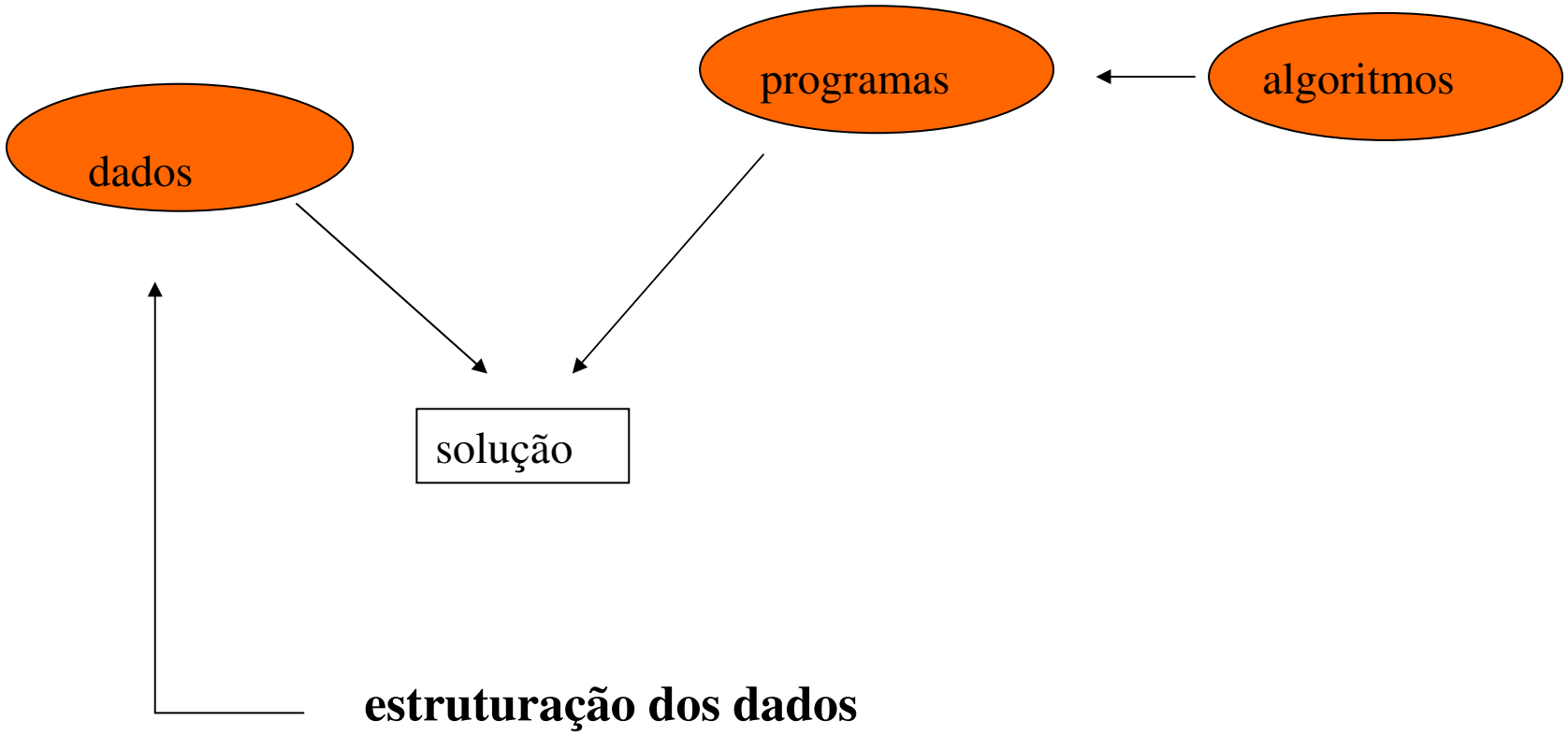
```
Nome(lista argumentos)
Declaração argumentos
{
  comandos
}
```

```
main ( )
{
 int i ;
for ( i=0; i<10; ++i)
printf (“%d%d%d \n”, i, power(2,i), power(-3,i));
}
power (x,n)
int x,n;
{
int i,p;
p=1;
 for ( i=1; i<=n; ++i)
p=p*x;
return(p);
}
```

1.7 Passos para resolver um problema



Passos para resolver um problema

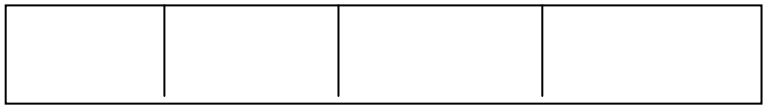


1.8 Estruturação dos dados

Dados podem ser estruturados de inúmeras formas em função da manipulação desejada.

Trataremos algumas destas formas (unidade de informação é denominada registro):

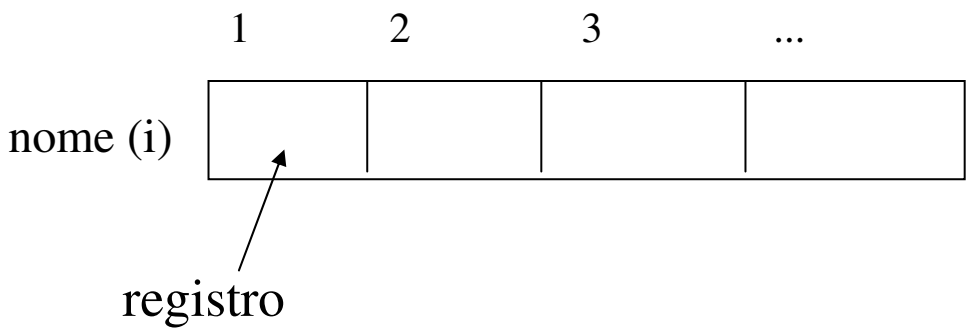
registro



estruturação de dados

- Vetor
- Fila
- Pilha
- Árvore

Vetor



Estrutura rígida, com todos os registros de mesmo tamanho. Não há a necessidade de demarcadores entre os registros. A ordem dos elementos é dada pelo índice.

Manipulação de forma aleatória com acesso direto a cada / todos elementos da estrutura.

Vetor - manipulação

O acesso aos elementos do vetor é realizado através do índice que fornece a posição do elemento a ser lido ou escrito. Com isto temos acesso aleatório e o tempo de acesso a qualquer elemento é idêntico.

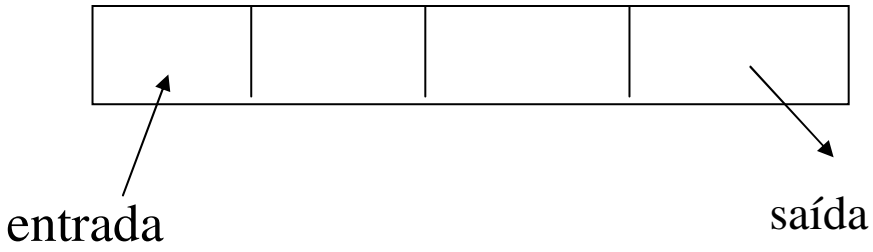
Leitura – de vetor (i)

- i é fornecido
- elemento ← vetor (i)

Escrita – em vetor (i)

- i é fornecido
- vetor (i) ← elemento

Fila



Estrutura: o acesso aos dados dá-se apenas nas posições “entrada” e “saída”, segundo a estratégia FIFO – First-In First-Out. As células são homogêneas.

Manipulação: somente as posições “entrada” e “saída” são visíveis.

Fila - manipulação

O acesso aos elementos da fila é realizado através das posições "entrada" e "saída", sendo que as outras posições não são acessíveis. Com isto temos acesso somente a estas duas posições da estrutura, que se movimentam conforme a estrutura cresce ou diminui.

Leitura – da posição "saída"

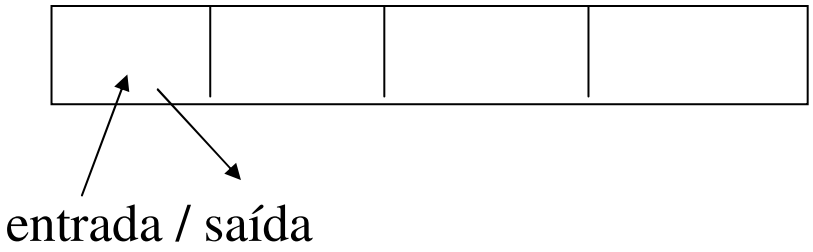
- elemento ← fila (saída)
- atualiza índice "saída"

Escrita – na posição "entrada"

- fila (entrada) ← elemento
- atualiza índice "entrada"

Obs.: necessidade de tratamento quando a estrutura estiver vazia ou cheia.

Pilha



Estrutura: o acesso aos dados dá-se apenas na posição "entrada / saída" segundo a estratégia LIFO – Last-In First-Out. As células são homogêneas.

Manipulação: somente a posição "entrada / saída" é visível.

Pilha - manipulação

O acesso aos elementos da pilha é realizado através da posição ES (“entrada / saída”), sendo que as outras posições não são acessíveis. Com isto temos acesso somente a esta posição da estrutura, que se movimenta conforme a estrutura cresce ou diminui.

Leitura – da posição ES

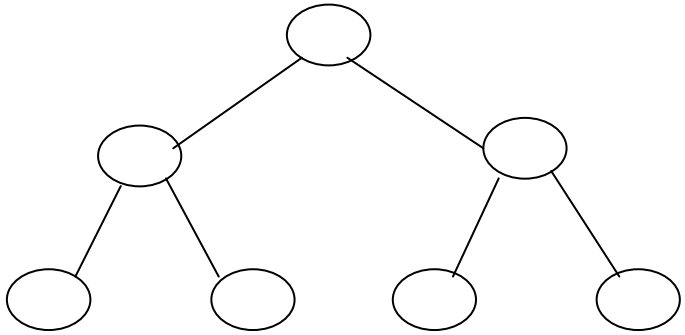
- atualiza índice “ES”
- elemento ← pilha (ES)

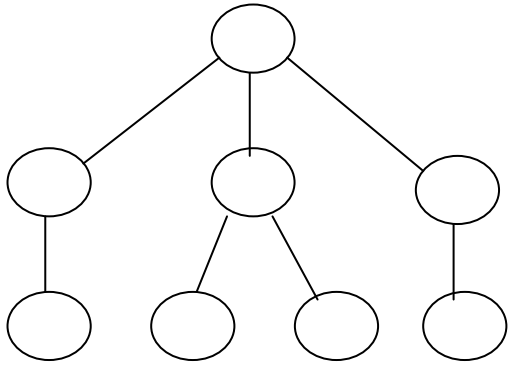
Escrita – na posição ES

- pilha (ES) ← elemento
- atualiza índice “ES”

Obs.: necessidade de tratamento quando a estrutura estiver vazia ou cheia.

Árvore





Estrutura: cada célula contém dado e apontador para a próxima célula. Células não necessitam ser homogêneas.

célula:

dados	apontador

Manipulação: flexível, com início pela raiz.

Árvore - manipulação

O acesso aos elementos da árvore é realizado percorrendo-se a mesma a partir de sua raiz e seguindo o(s) apontador(es) das células. Todas as posições da estrutura são acessíveis.

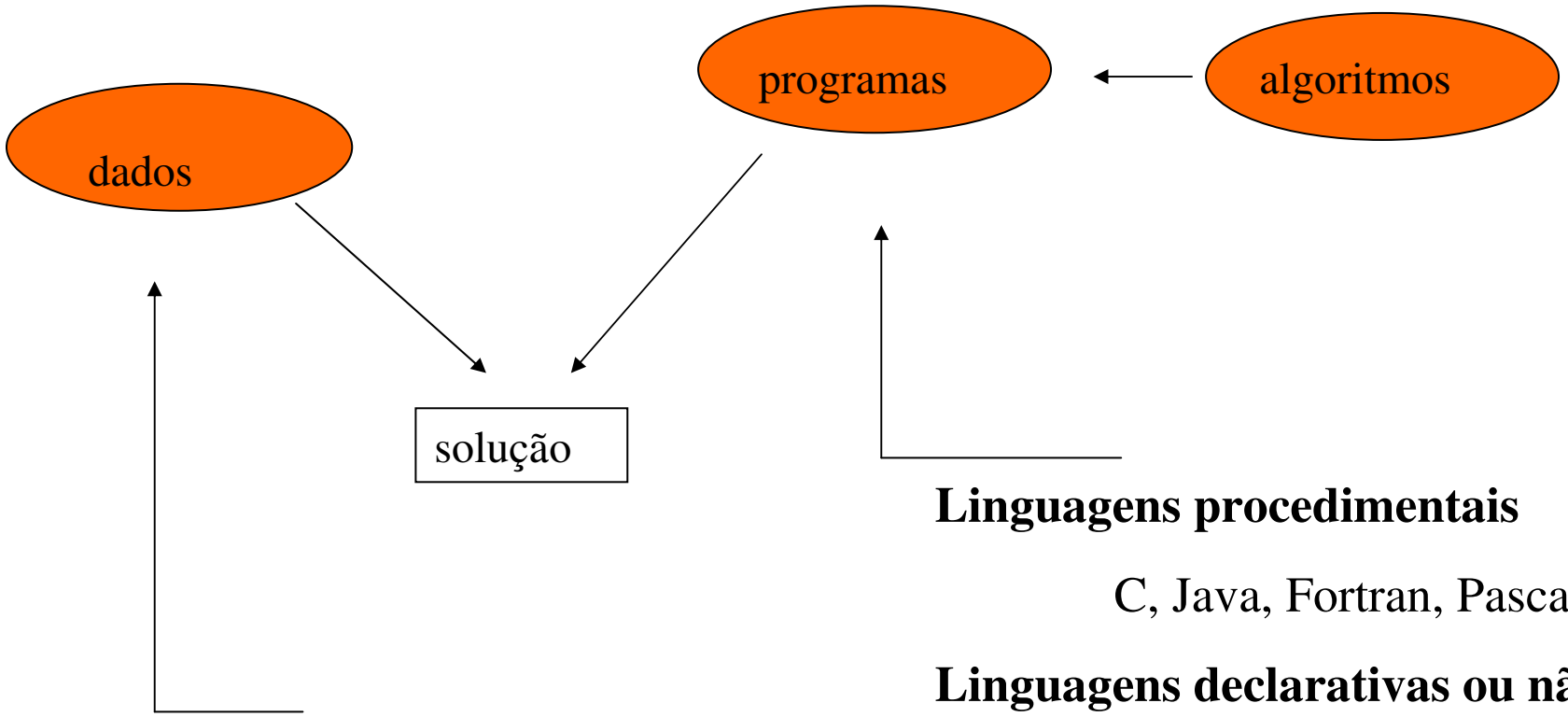
Leitura – da célula J

- navegar na estrutura até célula J
- elemento ← dados da célula

Escrita – na célula J

- navegar na estrutura até célula J
- dados da célula ← elemento

Passos para resolver um problema



Linguagens procedimentais

C, Java, Fortran, Pascal, ...

Linguagens declarativas ou não-procedimentais

Prolog, Lisp, ...

estruturação dos dados ✓

1.9 Linguagens

Linguagens Procedimentais

Os algoritmos são codificados em alguma linguagem procedimental, dependendo da natureza do problema e disponibilidades existentes.

Exemplo: programa para contar a qtde de dígitos, espaços, etc.

```
main() /*contagem de caracteres*/
{
    int c, i, nbranco, noutros, ndigitos[10];
    nbranco = noutros = 0;
    for (i = 0; i<10; ++i) ndigitos[i] = 0;
    while (( c = getchar ( ) ) != EOF)
      if (c >= "0" && c<= "9") ++ndigitos[c-"0"];
      else  if (c== " " || c== "\n" || c=="\t") ++nbranco;
      else ++ noutros;
    printf("dígitos = ")
    for (i = 0; i<10; ++i)
      printf("%d"n ndigitos[i]);
    printf(" \n espaços-brancos = %d, outros = %d \n", nbranco, noutros);
}
```

Linguagens declarativas (não-procedimentais

FATOS + REGRAS → **SOLUÇÃO**

João é filho de Paulo.

Maria é filha de Paulo.

+

Filhos de mesmo pai são irmãos.

Possível pergunta:

Maria e João são irmãos ?

Exemplos:

Declaração:

amiga (joana, maria)

amiga (maria, joana)

Questão:

? amiga (joana, maria).

Resposta: Sim.

Declaração:

amiga (joana, maria)

amiga (clara,maria)

Questão:

? amiga (joana, quem),

amiga (clara, quem).

Resposta: quem = maria.

amiga (joana, maria)

amiga (maria, joana)

amiga (maria, clara)

amiga (clara, maria)

ama (joana, maria)

ama (maria, joana)

def-regra: amiga_intima

amiga (Ela1, Ela2),

amiga (Ela2, Ela1),

ama (Ela1, Ela2)

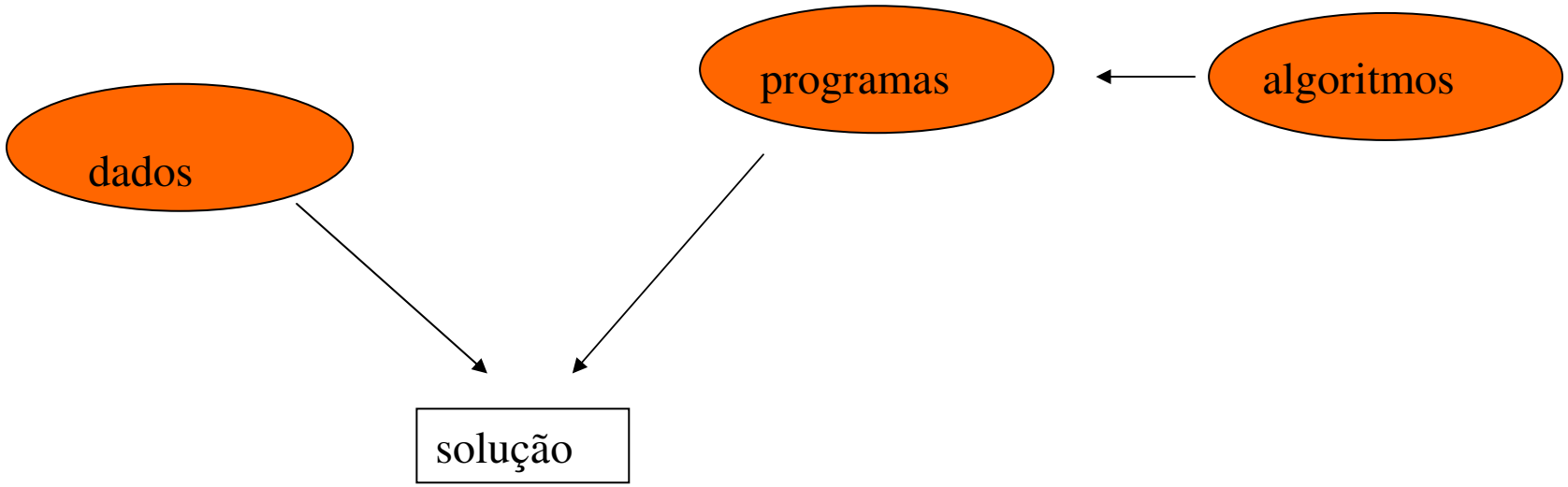
ama (Ela2, Ela1)

Questões:

? amiga_intima (maria, joana). Sim.

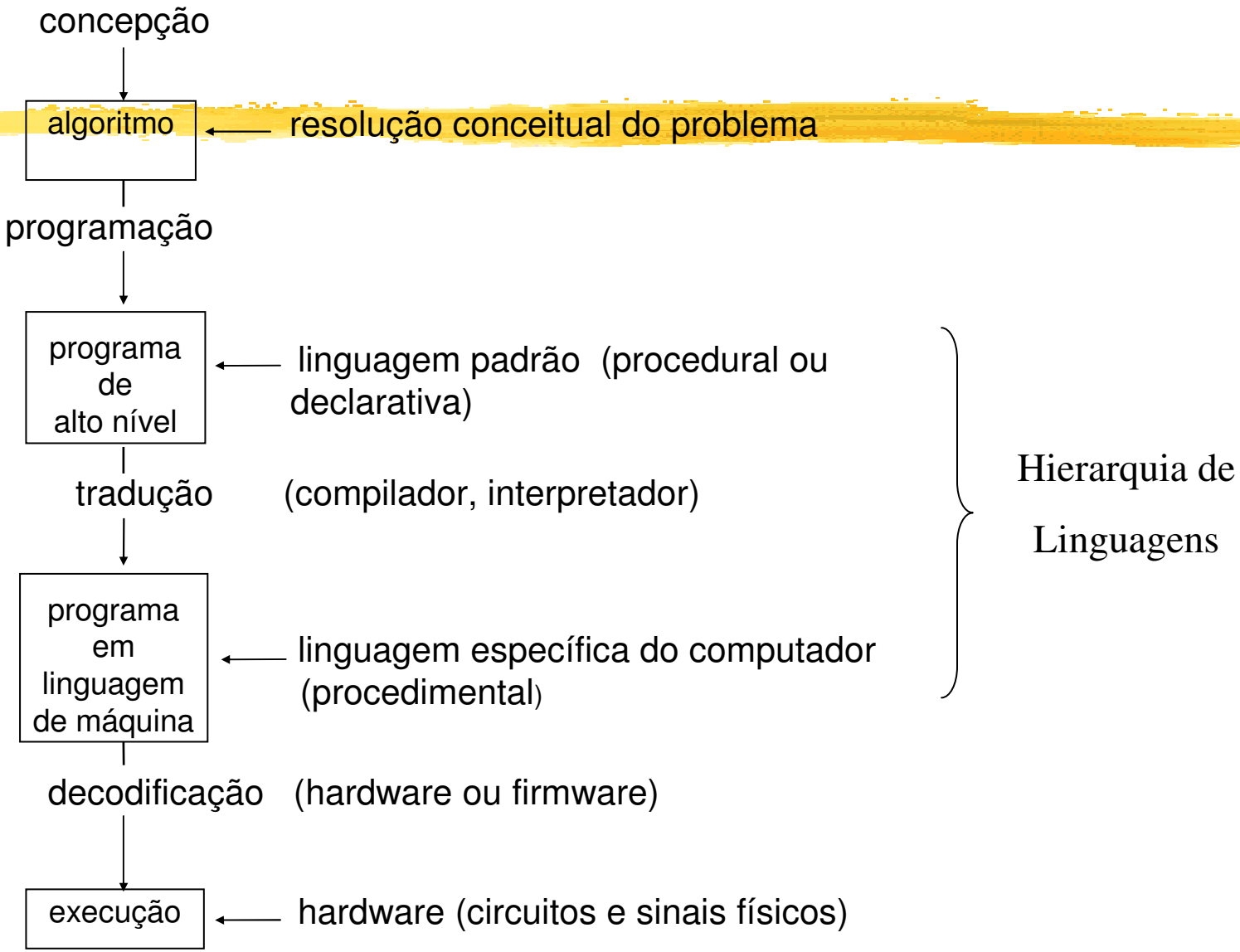
? amiga_intima (maria, clara). Não.

Voltemos à Figura:



Vamos tratá-la agora sob a perspectiva da execução em um sistema computacional da solução obtida.

Hierarquia da solução



Hierarquia de Linguagens

Linguagem

- de máquina
- simbólica
- de alto-nível
- voltada à aplicação

Hierarquia de Linguagens – cont.

Linguagem de máquina

Representação binária a ser processada pelo computador (que é uma máquina binária)

Exemplo: 0111011101110000

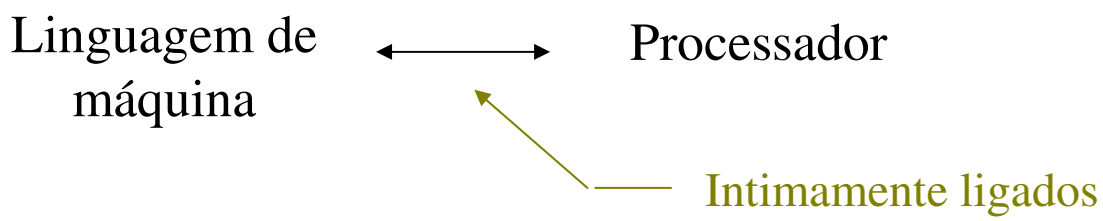
Linguagem de máquina ⟷ Processador

Hierarquia de Linguagens – cont.

Linguagem de máquina

Representação binária a ser processada pelo computador (que é uma máquina binária)

Exemplo: 0111011101110000



Hierarquia de Linguagens – cont.

Linguagem Simbólica

Representação simbólica dos passos a serem executados.

Embora não seja necessariamente específica de um processador está bem próxima (dependência) do mesmo.

Exemplo: MOVE R1, R2 ; R2 ← (R1)

tradução para o processador "X"

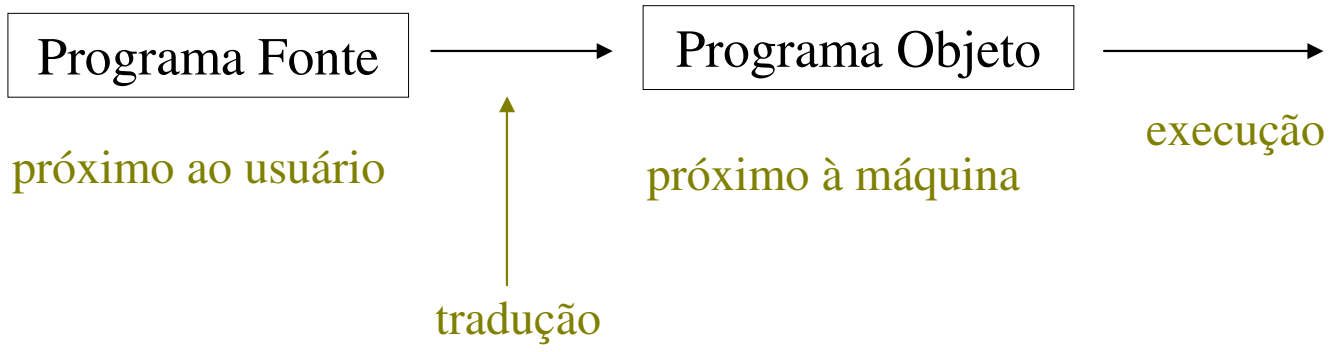
0111011101110000

Hierarquia de Linguagens – cont.

Linguagem de alto-nível

Estas linguagens (C, Java, C++, Pascal, ...) são assim denominadas por terem uma semântica próxima ao usuário.

Exemplo:



Hierarquia de Linguagens – cont.

Linguagens voltadas à aplicação

Como o próprio nome diz, estas linguagens estão intimamente ligadas à uma área de aplicação.

Exemplos: Matlab (arquivos .m), Access, etc.

Como se dá o processamento dos programas ?

Simplificadamente em uma primeira aproximação podemos pensar em 3 camadas:

- o hardware do computador propriamente dito, executando programas em linguagem de máquina;
- o software de sistema, contendo todos os programas de apoio ao uso da máquina
  - Editores, tradutores, carregadores, interface, acompanhamento da execução de programas, etc.
- Software de usuário com os programas voltados às aplicações de interesse (Matlab, por exemplo)

A máquina virtual – Linux, W95, W2000, XP, Vista, etc. – depende deste (Sotware de Sistema) nível.

1.10 Exercícios

1. Resolva o exercício 19 do livro texto (cap. 1).

2. Implemente, usando um sistema de seu conhecimento, todos os exemplos deste texto em linguagem C.

3. Considere as 4 formas de organização de dados tratadas: vetor, fila, pilha e árvore. Escreva para cada uma delas algoritmos para inserção, retirada (apagar) e leitura de dados, obedecendo as peculiaridades das mesmas. Implemente estes algoritmos em linguagem C.