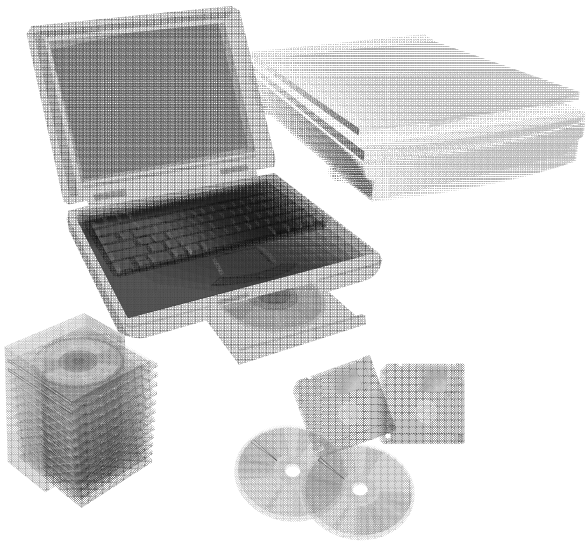
Introdução à Computação II



Modularização

Profa.: ***Adriane Beatriz de Souza Serapião***

adriane@rc.unesp.br

Programação

Para realizar um programa não basta dispor de uma linguagem de alto nível, temos de definir as ações a desenvolver para executar essa tarefa, caso contrário, não seremos capazes de realizar o programa na linguagem pretendida.

A concepção e implementação de bons programas é, para além de uma necessidade, um desafio.

Qualquer problema desenvolve-se em 4 fases :

a) Analise

b) Concepção

c) Implementação

d) Teste

Programação

Análise: **é a partir das informações recolhidas durante a análise que se especificam as funções que o programa deve cumprir.**

Concepção: **é delineado o esquema lógico a implementar no computador. Um bom esquema lógico permite a construção de programas eficientes, fáceis de corrigir e alterar.**

Implementação: **do esquema lógico, utilizando uma linguagem de programação. É nesta fase que se procede à escrita do texto do programa.**

Testes: **é nesta altura que o programa é testado, para se verificar se respeita integralmente as especificações resultantes da fase de análise.**

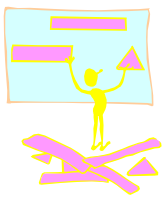
Programação

Definição de Algoritmos

Algoritmo é a seqüência de ações que devem ser determinadas para a obtenção da solução de um determinado problema.

Etapas na solução de problemas

- **Compreensão do problema;**
- **Criar uma seqüência de operações (ou ações) que quando executadas produzem a solução do problema;**
- **Execução da seqüência de operações.**



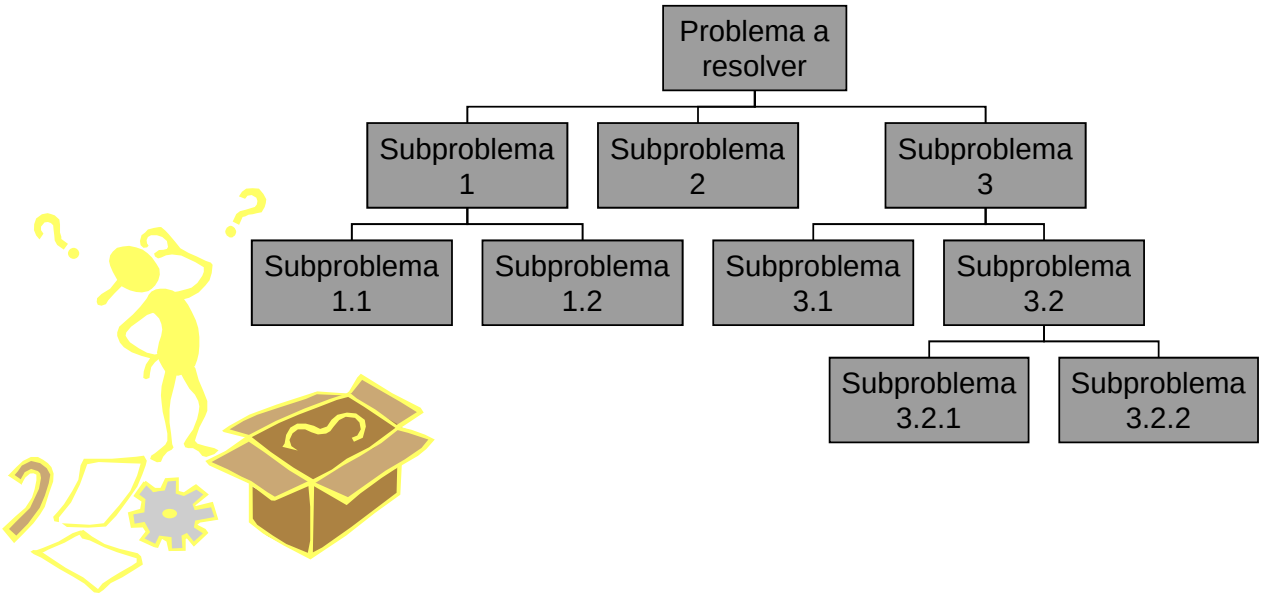
Propriedades dos algoritmos

- **Ações simples e bem definidas;**
- **Seqüência ordenada de ações;**
- **Seqüência finita de passos.**

Modularização

- **Estudos mostram que o ser humano consegue lidar bem com até 7 problemas ao mesmo tempo.**
- **A solução é o chamado "Método de Refinamentos sucessivos": dividir o problema em vários sub-problemas, evitando assim abordar todos os detalhes do problema simultaneamente.**
- **Nesse contexto, para cada sub-problema será criado uma subrotina, ou seja, um módulo.**
- **Vamos "Modularizar" o problema!**

Modularização



Modularização

- **Programação Top-Down (ou "de cima para baixo"):**

  1 - Inicialmente o programador deve saber as tarefas principais do programa, não como fazê-las, apenas quantificá-las;
  2 - Depois, modelar como o programa principal irá chamar (gerenciar) essas tarefas;
  3 - Então, cada tarefa é detalhada.

Modularização

- **Vantagens da Programação Top-Down:**
  - **reutilização de código fonte;**
  - **facilitar a compreensão de como o problema foi modelado;**
  - **a manutenção é feita por módulo;**
  - **alterações no módulo valem para todos os lugares onde o módulo é usado;**
  - **o número de linhas de um código fonte de um módulo é relativamente menor, e mais fácil para se entender;**
  - **etc.**

Modularização em Pascal

- **Uma subrotina é um sub-programa com variáveis e comandos próprios e que, para ser executada, precisa ser chamada pelo programa principal.**
- **Na linguagem PASCAL temos dois tipos de subrotinas:**
  - **Procedimentos (*procedures*);**
  - **Funções (*functions*).**
- **A função retorna um valor, o procedimento não.**

Procedimentos

- **Declaração**

```
void nome (lista-de-parâmetros)
        declaração de variáveis locais;
{
        comandos;
}
```

- **Exemplo:**

```
void Troca (float *A, float *B)
float aux;
{
        aux = A;
        A = B;
        B = aux;
 }
```

Procedimentos

```
// programa OrdemCrescente

// ------- SUBROTINA TROCA -------
void Troca (float *A, float *B)
float aux;
{
    aux=A;
    A=B;
    B=aux;
}
//  ------- FIM TROCA  -------

void main()
{
   float L,M,N;
    // ------- PROGRAMA PRINCIPAL -------
    scanf ("%f %f %f", &L, &M, &N);
    if ((L>M) || (L>N))
        if (M<N) Troca(&L, &M)
        else Troca(&L, &N);
    if (M>N) Troca(&M, &N);
    printf("%f %f %f", L, M, N);
    // ------- FIM PRINCIPAL  -------
}
```


Funções

- **Declaração**

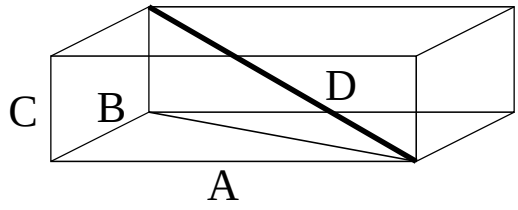
tipo **nome (lista-de-parâmetros)**
{
declaração de variáveis locais;
comandos;
}

- **Exemplo:**

```
float Hipotenusa (float A, float B)
{
   float H;
   H = sqrt( A*A + B*B );
   return H;
}
```

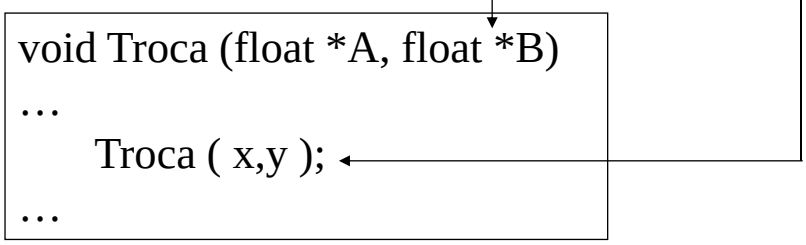
Funções

```
// programa Diagonal;
// Diagonal de um paralelepípedo

// Funcao Hipotenusa
float Hipotenusa (float A, float B)
{
   Hipotenusa:= sqrt ( sqr(A) + sqr(B) );
}
// Fim Funcao Hipotenusa

void main()
{
   float A, B, C, D;
   // Programa Principal
   scanf(%f %f %f %f", A, B, C, D); // dimensoes
   D = Hipotenusa ( Hipotenusa (A, B), C );
   printf("%f", D);
   // Fim Programa Principal
}
```



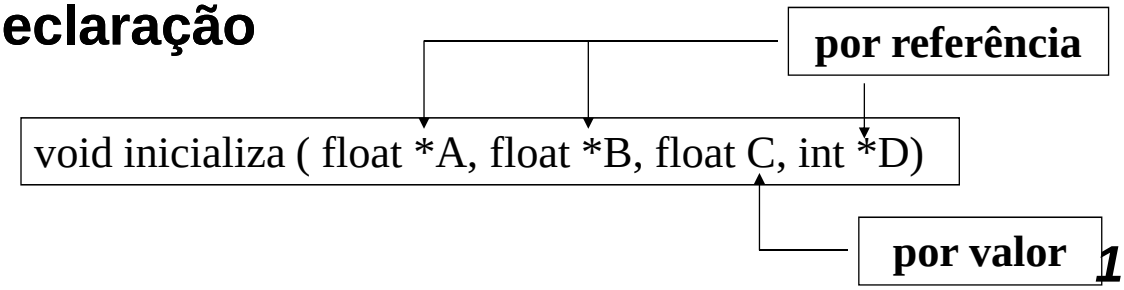
Nomenclatura dos parâmetros

- **Parâmetros**
  - **reais: chamadora;**
  - **formais: subrotina.**



Passagem de parâmetros

- **Passagem**
  - **por valor: Apenas o valor é transferido. Então, as alterações feitas nos parâmetros formais (da subrotina) não alteram os reais (chamadora).**
  - **por referência: O endereço do parâmetro real é transferido. Então, as alterações nos parâmetros formais da subrotina na verdade estão sendo feitas sobre os parâmetros reais.**
- **Declaração**



Passagem de parâmetros

```
// programa Parametros

void inicializa(float *A, float *B, float C; int *D)
{
   printf("Passo 1: %f %f %d %d\n", A, B, C, D);
   A=1; B=1; C=1; D=1;
   printf("Passo 2: %f %f %d %d\n", A, B, C, D);
}

void main()
{
   float X,Y,Z;
   int W;

   X=0; Y=0; Z=0; W=0;
   inicializa (X, Y, Z, W);
   printf ("Passo 3: %f %f %f %f", X, Y, Z, W);
}
```