

Subprogramação

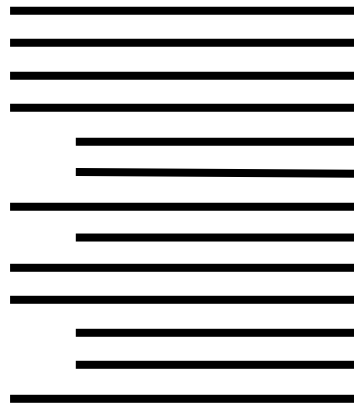
Leonardo Gresta Paulino Murta
leomurta@ic.uff.br

Aula de hoje

- Estudaremos duas estruturas de encapsulamento
 - Procedimentos
 - Funções

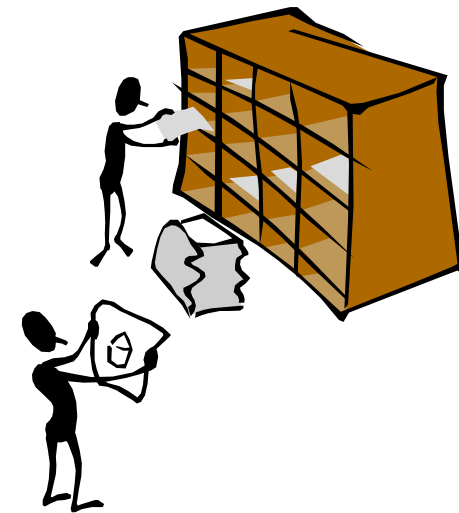
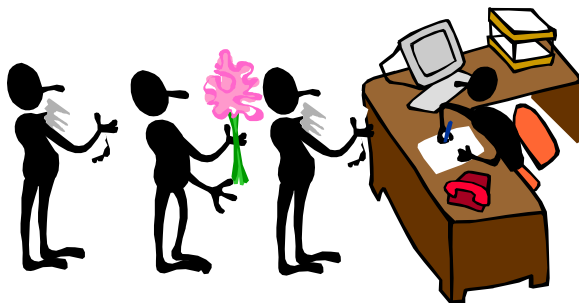
Retomando: paradigma estruturado

- Código mais fácil de ler, mas **ainda difícil para sistemas grandes** devido a repetição de código
 - Só usa sequência, repetição e decisão
- O que fazer se for necessário **repetir uma sequência de linhas de código** em diferentes locais?



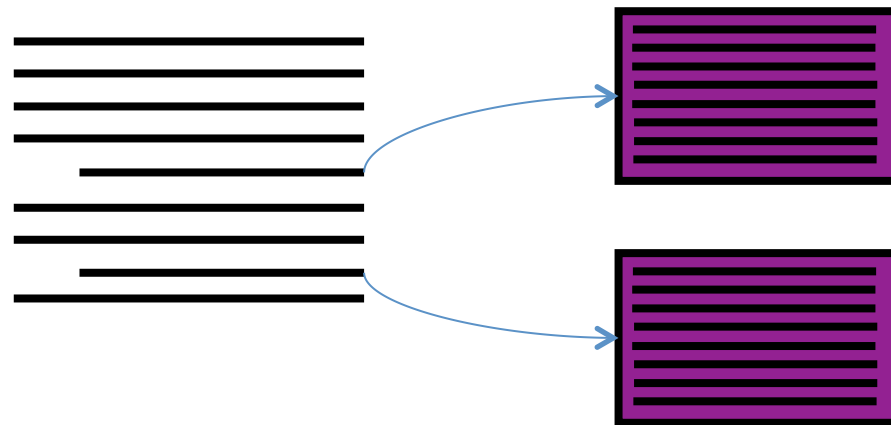
Encapsulamento

- Mecanismo utilizado para lidar com o aumento de complexidade
- Consiste em exibir “o que” pode ser feito sem informar “como” é feito
- Permite que a granularidade de abstração do sistema seja alterada, criando estruturas mais abstratas



Paradigma procedimental

- Sinônimo: paradigma procedural
- Uso de subprogramação
 - Agrupamento de código permitindo a criação de ações complexas
 - Atribuição de um nome para essas ações complexas
 - Chamada a essas ações complexas de qualquer ponto do programa
- Em Pascal, essas ações complexas são denominadas procedimentos ou funções



Exemplo

```
program calculo_imc;
```

```
var
```

```
  altura, massa, imc : real;
```

```
begin
```

```
  write('Entre com a sua altura em metros: ');  
  readln(altura);
```

```
  write('Entre com a sua massa em kg: ');  
  readln(massa);
```

```
  imc := massa / sqr(altura);  
  writeln('Seu IMC é ', imc);
```

```
end.
```

Parecidos!



Exemplo usando procedimento

```
program calculo_imc;
var altura, massa, imc : real;
```

Declaração do
procedimento

```
procedure leia(mensagem : string; var valor : real);
begin
  write(mensagem);
  readln(valor);
end;
```

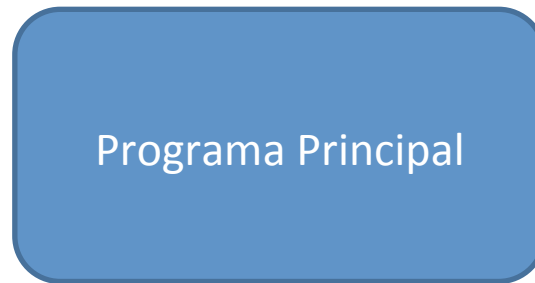
```
begin
  leia('Entre com a sua altura em metros: ', altura);
  leia('Entre com a sua massa em kg: ', massa);
```

```
  imc := massa / sqr(altura);
  writeln('Seu IMC é ', imc);
end.
```

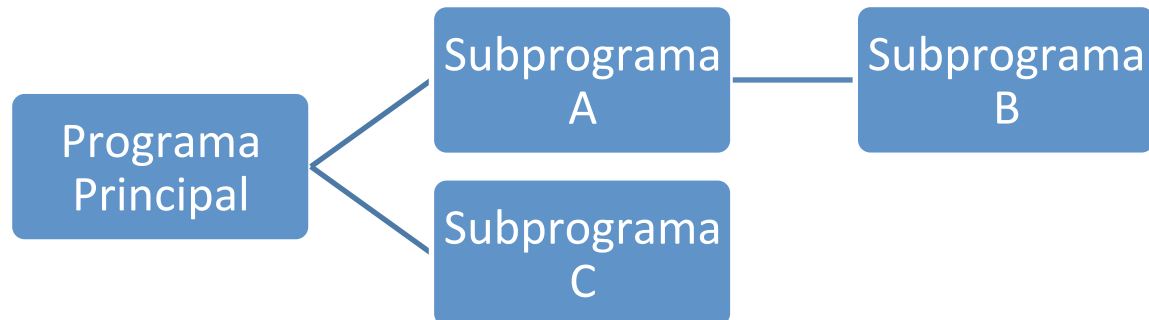
Chamadas ao
procedimento

Dividir para conquistar

- Antes: um programa gigante



- Depois: vários programas menores



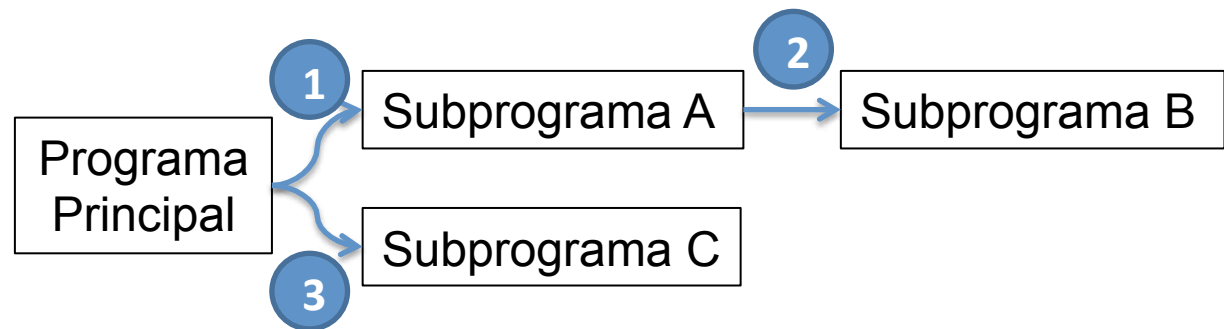
Fluxo de execução

- A execução inicia pelo programa principal
- O programa principal chama subprogramas
- Estes subprogramas podem chamar outros subprogramas, sucessivamente
- Ao fim da execução de um subprograma, a execução continua na instrução seguinte à da chamada do subprograma

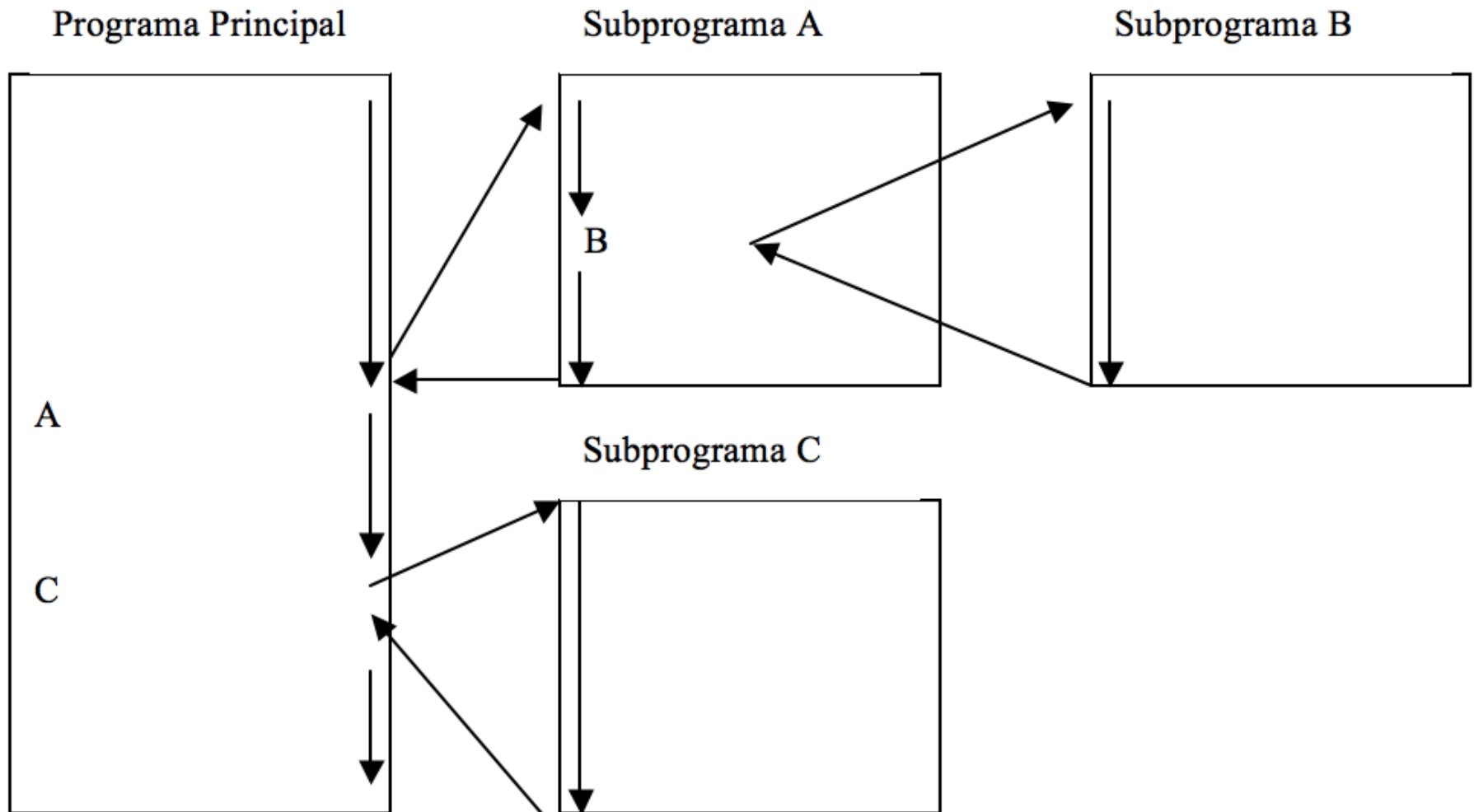
Programa

Subprograma C
Subprograma B
Subprograma A
Programa Principal

Possível sequencia de chamadas

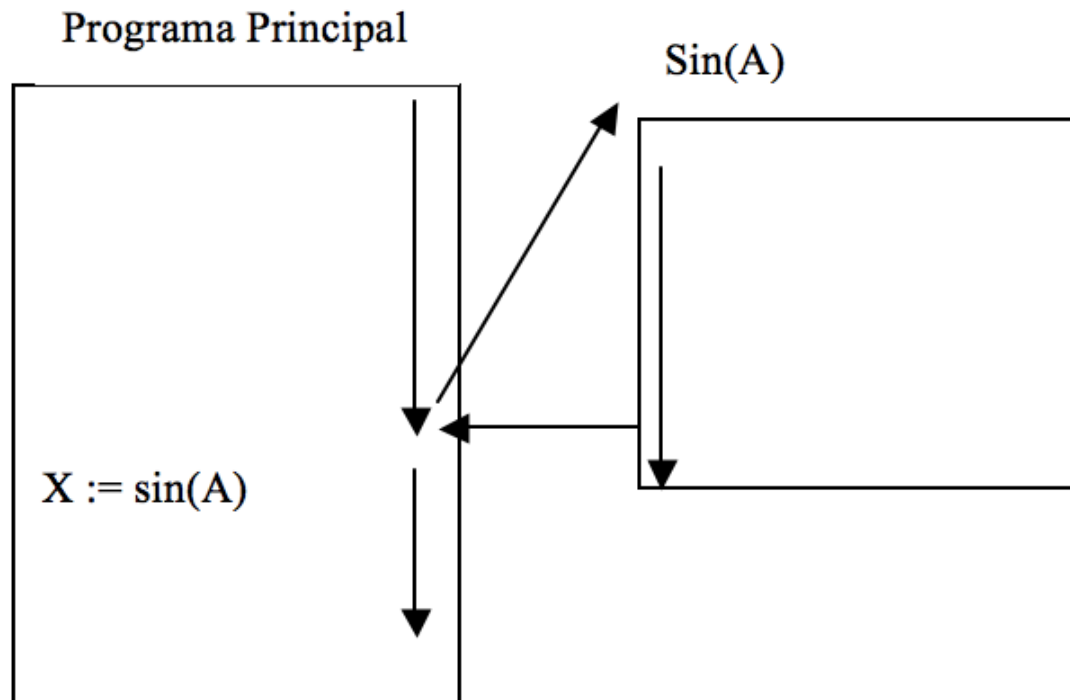


Fluxo de execução



Fluxo de execução

- É equivalente ao que acontece quando chamamos uma função predefinida do Pascal

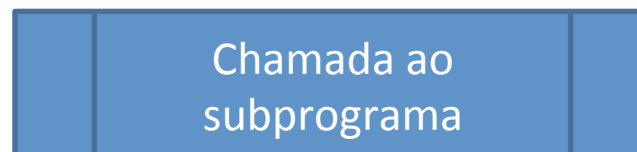


Vantagens

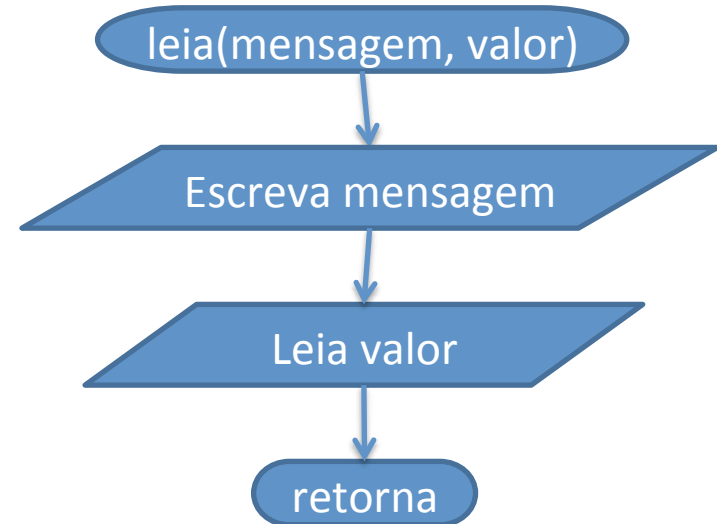
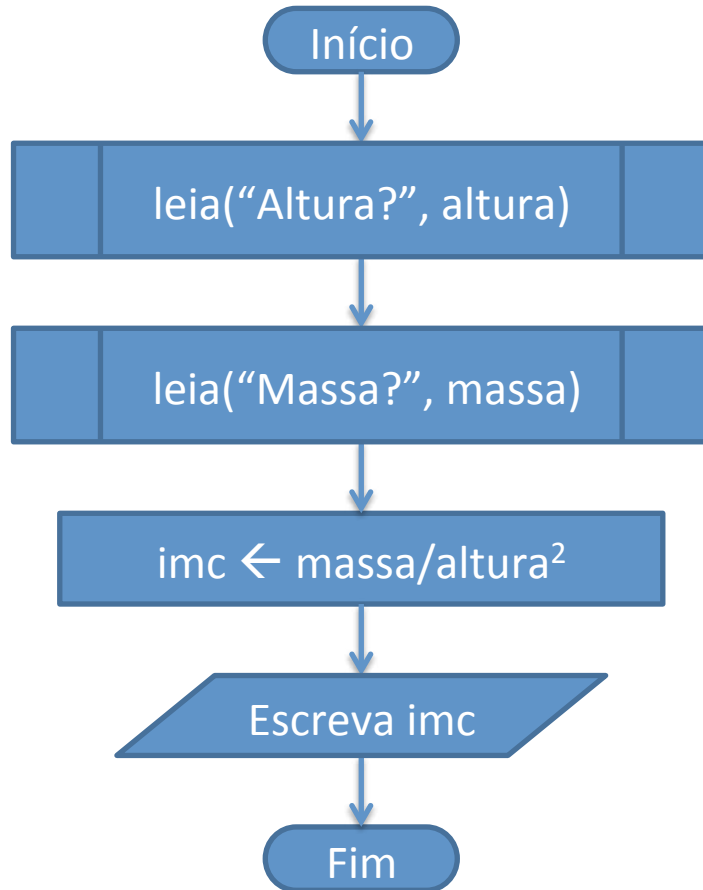
- Economia de código
 - Quanto mais repetição, mais economia
- Facilidade na correção de defeitos
 - Corrigir o defeito em um único local
- Legibilidade do código
 - Podemos dar nomes mais intuitivos a blocos de código
 - É como se criássemos nossos próprios comandos
- Melhor tratamento de complexidade
 - Estratégia de “dividir para conquistar” nos permite lidar melhor com a complexidade de programas grandes
 - Abordagem *top-down* ajuda a pensar!

Fluxograma

- O subprograma passa a ter um fluxograma próprio, com o símbolo de início contendo o nome do subprograma e o símbolo de término contendo o retorno
- Além disso, um novo símbolo é utilizado no programa principal para indicar a chamada a um subprograma:



Exemplo de Fluxograma



Tipos de Subprogramas

- Procedimento
 - Não tem valor de retorno
- Função
 - Deve retornar um e somente um valor
 - Esse valor retornado pode ser composto (vetor, matriz ou registro)

Estrutura de um programa com subprogramas

```

program NOME DO PROGRAMA;
var ...

procedure ...
var ...
begin
    CÓDIGO DO SUBPROGRAMA
end;
...
function ...
var ...
begin
    CÓDIGO DO SUBPROGRAMA
end;

begin
    CÓDIGO DO PROGRAMA
end.

```


Chamadas entre subprogramas

- Um subprograma A só é capaz de chamar um subprograma B se a declaração de B vier antes da declaração de A

```

procedure b;
begin
    ...
end;

...
procedure a;
begin
    b;
end;
    
```

Chamadas entre subprogramas

- Se for necessário chamar um subprograma que vem depois, é preciso colocar a declaração seguida da palavra *forward*
- Evitem dependência circular entre subprogramas

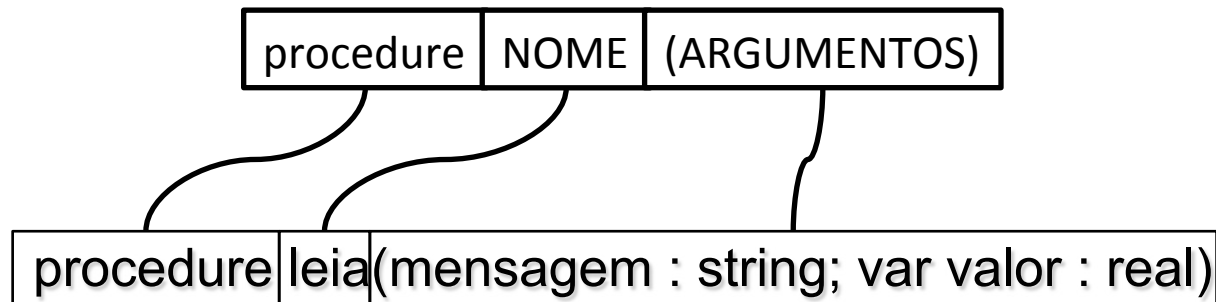
```
procedure b; forward;
```

```
procedure a;
begin
  b;
end;
```

...

```
procedure b;
begin
  ...
end;
```

Sintaxe da declaração de um procedimento



Argumentos

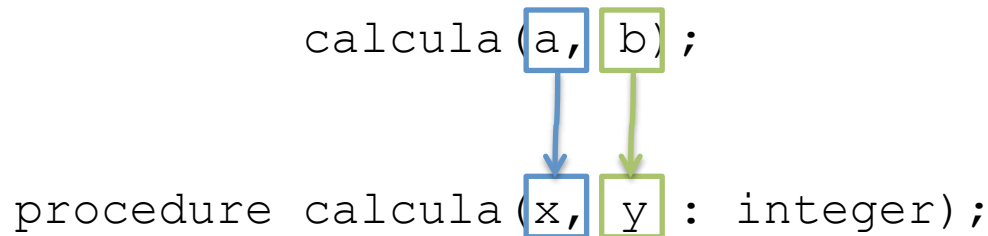
- Declarados de forma equivalente à declaração de variáveis de um programa
 - Separados por “;”
- A passagem de argumentos do subprograma é posicional (1º parâmetro = 1º argumento, 2º parâmetro = 2º argumento, etc.)
- Há duas formas de passagem de dados
 - Por valor
 - Por referência

Passagem de argumento por valor

- A alteração da variável não será visível fora do escopo do subprograma
- Não tem “var” antes da declaração do argumento
 - Exemplo: NOMES : TIPO
- O Pascal **copiará o valor** de cada argumento para a respectiva variável na ordem que aparecem

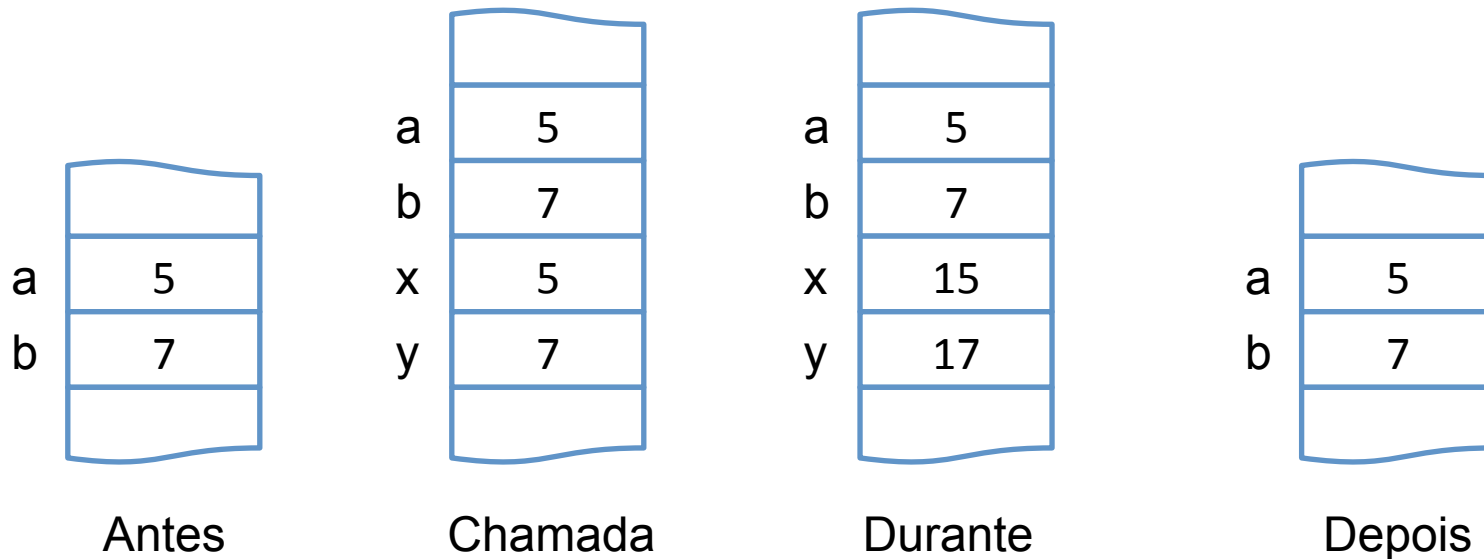
```

calcula (a, b);
          | |
          v v
procedure calcula (x, y : integer);
  
```



Passagem de argumento por valor

- Na memória:

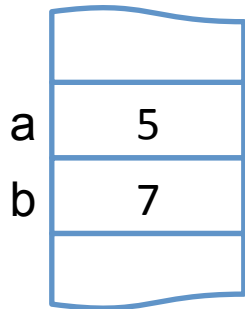


```

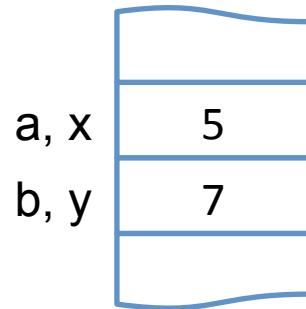
calcula(a, b);
          |  |
          v  v
procedure calcula(x, y : integer);
  
```


Passagem de argumento por referência

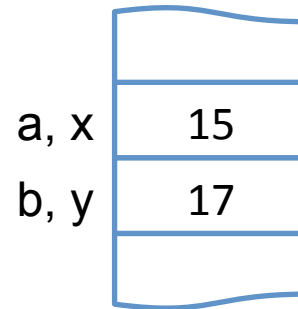
- Na memória:



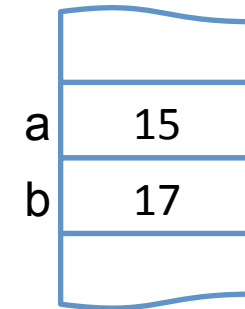
Antes



Chamada



Durante



Depois

calcula(a, b);

procedure calcula(var x, y : integer);

Argumentos do tipo array

- Não é permitido declarar argumento do tipo array de forma implícita
- Quando for usar um argumento do tipo array, é necessário criar um tipo composto (usando type) antes

```

type tabuleiro = array[1..3, 1..3] of char;
...
procedure desenha(tab : tabuleiro);
begin
...
end;

```

Variáveis globais e locais

- As variáveis declaradas no programa principal, antes da declaração dos subprogramas, podem ser usadas em todos os subprogramas
 - São conhecidas como **variáveis globais**
 - Tentem evitar usar variáveis globais (deem preferência por passar os valores via argumentos)
- Um subprograma pode criar suas próprias variáveis
 - São conhecidas como **variáveis locais**
 - Assim como no programa principal a declaração de variáveis locais deve ser feita logo antes do begin do subprograma
 - As variáveis locais e os argumentos são válidos somente no escopo do subprograma (dentro do begin/end do subprograma)

Refazendo o exemplo com função...

```
program calculo_imc;
var altura, massa, imc : real;
```

```
function leia(mensagem : string) : real;
begin
  write(mensagem);
  readln(leia);
end;
```

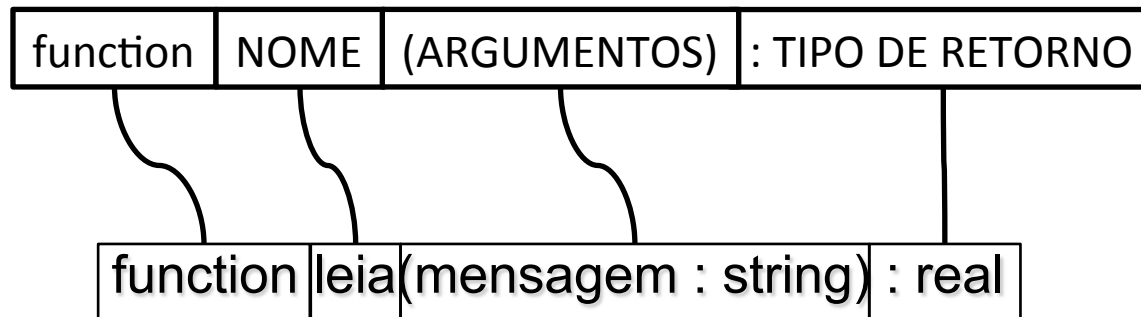
Declaração da
função

```
begin
  altura := leia('Entre com a sua altura em metros: ');
  massa := leia('Entre com a sua massa em kg: ');

  imc := massa / sqr(altura);
  writeln('Seu IMC é ', imc);
end.
```

Chamadas à
função

Sintaxe da declaração de uma função



Retorno da Função

- O Pascal cria automaticamente uma variável com o nome da função
- O valor a ser retornado pela função deve ser atribuído a essa variável

Exemplo

```

program troca;

var a, b : integer;

procedure troca(var x, y : integer);
var aux : integer;
begin
    aux := x;
    x := y;
    y := aux;
end;

function media(x, y : integer) : real;
begin
    media := (x + y) / 2;
end;

begin
    a := 5;
    b := 7;
    troca(a, b);
    writeln('a: ', a, ', b: ', b);
    writeln('média: ', media(a,b));
end.

```

Sobrecarga de subprograma

- Uma programa pode ter **dois ou mais subprogramas com o mesmo nome** desde que os tipos de seus argumentos sejam distintos
- Isso é útil quando queremos implementar um subprograma em função de outro
- Exemplo:

```
function metro2pes(x : real) : real;
begin
    ...
end;
```

```
function metro2pes : real;
begin
    metro2pes := metro2pes(1);
end;
```

Subprogramas sem argumentos

- Não é necessário ter argumentos nos subprogramas
 - Nestes casos, é opcional ter () depois do nome do subprograma
 - A chamada ao subprograma pode ou não conter ()

- Exemplo de declaração:

```
procedure pulaLinha;
begin
  writeln;
end;
```

- Exemplo de chamada:

```
pulaLinha(); OU pulalinha;
```


Exercício

- Faça uma função que informe o status do aluno a partir da sua média de acordo com a tabela a seguir:
 - Nota acima de 6 → “Aprovado”
 - Nota entre 4 e 6 → “Verificação Suplementar”
 - Nota abaixo de 4 → “Reprovado”
- O professor deseja dividir uma turma com N alunos em dois grupos: um com M alunos e outro com (N-M) alunos. Faça o programa que lê o valor de N e M e informa o número de combinações possíveis
 - Número de combinações é igual a $N! / (M! * (N-M)!)$
- Faça o programa de cálculo da série de Fibonacci usando um subprograma

Exercício

- Faça uma calculadora que forneça as seguintes opções para o usuário, usando subprogramas sempre que possível

Estado da memória: 0

Opções:

- (1) Somar
- (2) Subtrair
- (3) Multiplicar
- (4) Dividir
- (5) Limpar memória
- (6) Sair do programa

Qual opção você deseja?

Exercício

- Ordene um vetor de números inteiros com 10 posições utilizando os algoritmos
 - Bubble Sort
 - Selection Sort
 - Insertion Sort

Subprogramação

Leonardo Gresta Paulino Murta
leomurta@ic.uff.br