Sub-algoritmos: Funções e Procedimentos

- Nesta aula veremos o conceito de sub-algoritmo (ou sub-rotina): funções e procedimentos
- Sub-algoritmos são blocos de instruções que realizam tarefas específicas
- O código de um sub-algoritmo é carregado uma vez e pode ser executado quantas vezes for necessário
- Assim, os programas tendem a ficar menores e mais organizados, uma vez que o problema pode ser dividido em tarefas menores

José Augusto Baranauskas
Departamento de Física e Matemática – FFCLRP-USP
Sala 226 – Bloco P2 – Fone (16) 3602-4361
E-mail: augusto@ffclrp.usp.br
URL: http://fmrp.usp.br/augusto

Sub-algoritmos

- Em geral, um programa é executado linearmente, uma linha após a outra, até o fim
- Entretanto, quando são utilizados sub-algoritmos, é possível a realização de desvios na execução natural dos programas
- Assim, um programa é executado linearmente até a chamada de um sub-algoritmo
- O programa que chama um sub-algoritmo ("chamador") é temporariamente suspenso e o controle é passado para o sub-algoritmo, que é executado
- Ao terminar o sub-algoritmo, o controle retorna para o programa que realizou a chamada ("chamador")
- Tipos de Sub-algoritmos:
  - Funções (*functions*)
  - Procedimentos (*procedures*)

Funções

- É comum encontrar-se nas linguagens de programação, várias funções embutidas, por exemplo, **sin** (seno), **cos** (co-seno), **abs** (valor absoluto), **sqrt** (raiz quadrada)
- Funções embutidas podem ser utilizadas diretamente em expressões
- Por exemplo, o comando:
  - hipotenusa ← sqrt(cateto1^2 + cateto2^2)
  - calcula a hipotenusa de um triângulo retângulo como a raiz quadrada da soma dos quadrados dos dois catetos

Funções

- Essas funções são utilizadas em expressões como se fossem *simplesmente* variáveis comuns
- Como variáveis comuns, as funções têm (ou *retornam*) **um único valor**
- É responsabilidade do programador fornecer os *argumentos* (ou *parâmetros*) necessários para a função efetuar seus cálculos
- Por exemplo
  - A função **abs** tem como parâmetro um número, retornando um valor numérico
  - Em C/C++, a função **pow** tem como parâmetros dois números, retornando um valor numérico
  - Em C/C++, a função **strlen** tem como parâmetro uma string, retornando um valor numérico inteiro

Funções

- A utilização de funções afeta o fluxo de controle em um programa
- Quando uma função é chamada, o programa que chama a função fica em estado de espera e o controle passa para as instruções que definem a função
- Após a execução da função com os parâmetros fornecidos, o controle retorna ao ponto de chamada da função, com o valor calculado pela função

Função: Fluxo de Controle

```
Algoritmo Exemplo
Início
   declare x,y,valor : real

   x ← -10
   y ← 16
   valor ← abs(x) + sqrt(y)
   Escreva(valor)
Fim
```

Memória	
Endereço	Valor

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	
y	
valor	

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	-10
y	
valor	

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	-10
y	16
valor	

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	-10
y	16
valor	

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

O controle é transferido para a função **abs**

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	-10
y	16
valor	

Função **abs** — Instruções definindo a função **abs**

Função **sqrt** — Instruções definindo a função **sqrt**

Função: Fluxo de Controle

O controle é transferido para a função **abs**

```
Algoritmo Exemplo
Inicio
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

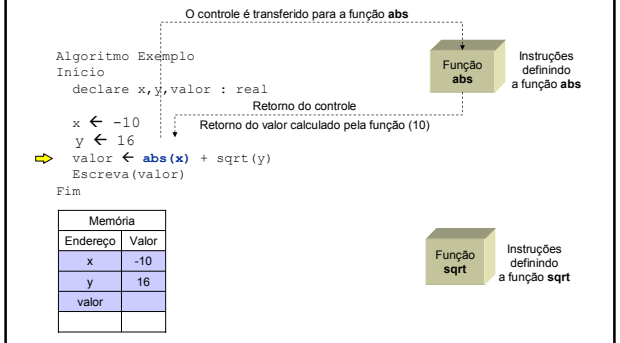
Memória	
Endereço	Valor
x	-10
y	16
valor	

Função **abs** — Instruções definindo a função **abs**

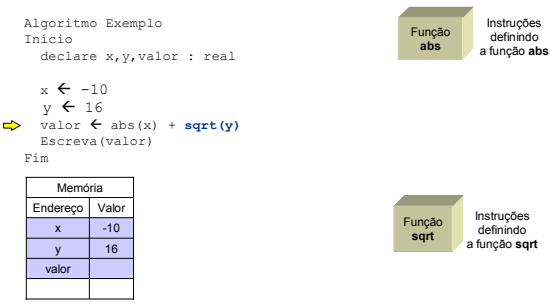
O código que define a função é executado, respeitando as estruturas de controle

Função **sqrt** — Instruções definindo a função **sqrt**

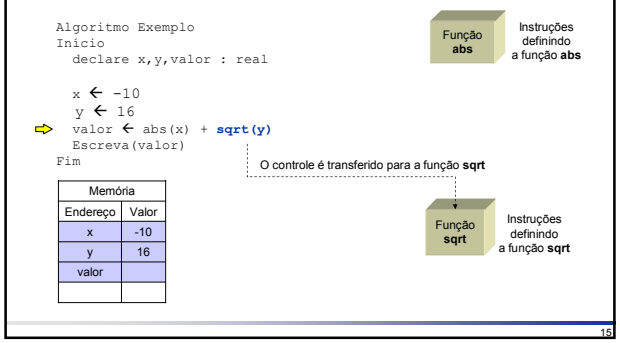
Função: Fluxo de Controle



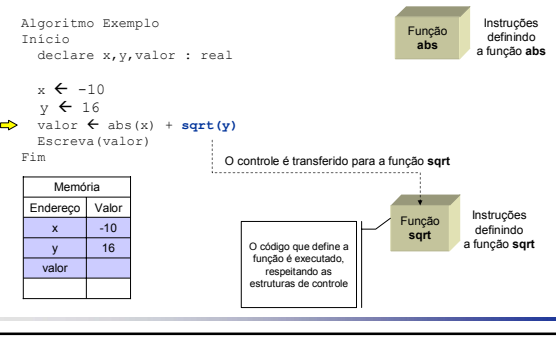
Função: Fluxo de Controle



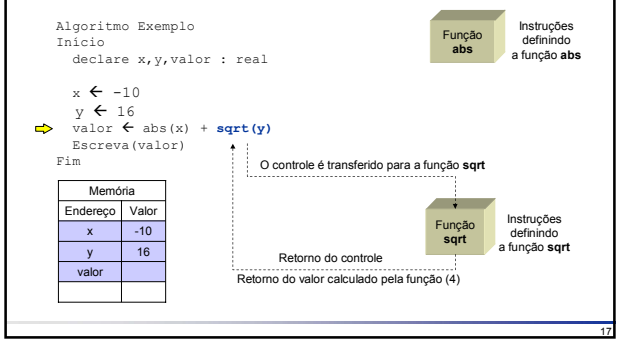
Função: Fluxo de Controle



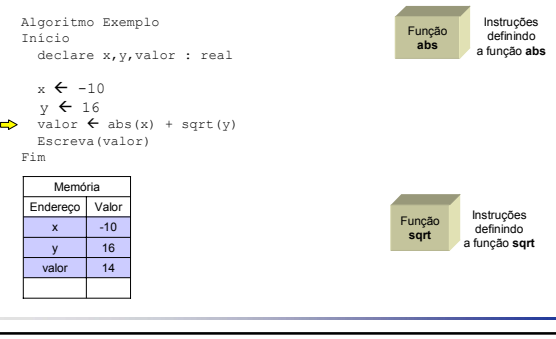
Função: Fluxo de Controle



Função: Fluxo de Controle



Função: Fluxo de Controle



Função: Fluxo de Controle

```
Algoritmo Exemplo
Início
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor
x	-10
y	16
valor	14

Instruções definindo a função **abs**

Instruções definindo a função **sqrt**

Função: Fluxo de Controle

```
Algoritmo Exemplo
Início
  declare x,y,valor : real

  x ← -10
  y ← 16
  valor ← abs(x) + sqrt(y)
  Escreva(valor)
Fim
```

Memória	
Endereço	Valor

Instruções definindo a função **abs**

Instruções definindo a função **sqrt**

Funções

- Em algumas situações, o programador gostaria de utilizar (definir) novas funções
- Por analogia, na Matemática, escreve-se (ou *define-se*) uma função por meio de parâmetros, por exemplo:
  - $f(x) = x^2 - 3x + 2$ *Definição da função f*
  - Esta função *f* foi definida em termos do parâmetro *x*
- Para saber o valor da função para um valor particular do argumento *x*, por exemplo, *x* = 3, basta substituir este valor onde aparece o parâmetro *x*:
  - $f(3) = 3^2 - 3(3) + 2 = 2$ *"Ativação" da função*
  - $f(1) = 1^2 - 3(1) + 2 = 0$
  - $f(-1) = (-1)^2 - 3(-1) + 2 = 6$
- Uma vez definida a nova função, ela pode ser utilizada sempre que necessária, mesmo em outras (novas) funções

Funções

- Como na Matemática, os parâmetros podem ser nomeados livremente
  - Por exemplo, são equivalentes as funções
    - $f(x) = x^2 - 3x + 2$
    - $f(y) = y^2 - 3y + 2$
- O nome da função é definido pelo programador e segue a mesma norma de formação de identificadores
- Por exemplo, são equivalentes as funções
  - $f(x) = x^2 - 3x + 2$
  - $g(x) = x^2 - 3x + 2$
  - $f(y) = y^2 - 3y + 2$
  - $g(y) = y^2 - 3y + 2$

Funções

- Funções podem ter mais de um parâmetro (argumento):
  - $g(x,y) = x^2 + y^3$
    - **g** possui 2 parâmetros
  - $h(x,y,z) = x^2 + 2y + z^2$
    - **h** possui 3 parâmetros
- Pode-se avaliar cada uma dessas funções de forma análoga:
  - $g(3,2) = 3^2 + 2^3 = 9 + 8 = 17$
  - $h(1,3,2) = 1^2 + 2(3) + 2^2 = 1 + 6 + 4 = 11$
- Notar a correspondência estabelecida entre os parâmetros da definição da função e os parâmetros de ativação (ou execução) da função
- No caso da função *g*, 3 é substituído para cada ocorrência de *x* e 2 é substituído para cada ocorrência de *y*. Essa ordem é fundamental, pois *g(3,2)* **não** é o mesmo que *g(2,3)*

Exemplo

```
//-------------------------------------------------------------
// Encontra o máximo entre dois inteiros
Função Maximo(x,y : inteiro) : inteiro
  declare max : inteiro

  Se (x > y) Então
    max ← x
  Senão
    max ← y
  Fim Se
  Retorne(max)
Fim Função
//-------------------------------------------------------------
Algoritmo TesteMaximo
Início
  declare a,b : inteiro

  a ← 5
  b ← 10
  Escreva("Máximo de ",a," e ",b,"=",Maximo(a,b))
  Escreva("Máximo de ",a+10," e ",b-5,"=",Maximo(a+10,b-5))
Fim
```

Outro Exemplo

```
//------------------------------------------------------------
// Encontra o máximo entre dois inteiros
Função Maximo(x,y : inteiro) : inteiro
  declare max : inteiro

  Se (x > y) Então
    max ← x
  Senão
    max ← y
  Fim Se
  Retorne(max)
Fim Função
//------------------------------------------------------------
// Encontra o máximo entre tres inteiros
Função Maximo3(a,b,c : inteiro) : inteiro
  Retorne(Maximo(a,Maximo(b,c)))
Fim Função
//------------------------------------------------------------
Algoritmo TesteMaximo
Inicio
  declare a,b,c : inteiro

  a ← 5
  b ← 10
  c ← 15
  Escreva(Maximo3(a,b,c))
Fim
```


Sub-algoritmos em C++

- A declaração (definição) de um sub-algoritmo (função ou procedimento) em C++ é similar à do programa principal (**main**)
- Sub-algoritmos preferencialmente devem ser declarados antes do programa que os chama
- É permitido declarar variáveis dentro do sub-algoritmo
  - As variáveis declaradas dentro do sub-algoritmo, incluindo os parâmetros são denominadas **variáveis locais**

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```


Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```

Indica o tipo de dado que a função deve retornar. Neste caso, a função retorna um número real

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```

Indica o nome da função. Neste caso, f

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```

Indica o tipo de dado do primeiro parâmetro da função. Neste caso é um número real

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```

Indica o nome do primeiro parâmetro da função. Neste caso, x

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{
  return(x * x - 3 * x + 2);
}
```

Indica o que deve ser retornado pela função

Funções em C++

- Função $f(x) = x^2 - 3x + 2$

```
float f(float x)
{ float r;

  r = x * x - 3 * x + 2;
  return r;
}
```

Se houver necessidade, variáveis (locais) adicionais podem ser declaradas dentro da função, de forma análoga à declaração de variáveis no programa principal

Exemplo de Chamada em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------------
float f(float x)              /* definicao da funcao f */
{
  return x * x - 3 * x + 2;
}
//-------------------------------------------------------------
int main()                    /* programa principal */
{ float a,r;

  a = 3;
  /* utilizar a funcao numa expressao simples */
  r = f(a);                                  /* ativacao de f */
  cout << "Valor de f(" << a << ")=" << r << endl;
  /* utilizar a funcao numa expressao mais elaborada */
  r = f(2*a) + 3 * f(1) + 2 * f(-1); /* ativacao de f */
  cout << r << endl;
  return 0;
}
```


Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	3
divisor	4

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	3
divisor	4
quociente	

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	3
divisor	4
quociente	0.75

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

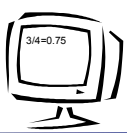
```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	3
divisor	4
quociente	0.75

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```



Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	4
divisor	3

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	4
divisor	3
quociente	

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	4
divisor	3
quociente	1.33

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	4
divisor	3
quociente	1.33

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	3
b	4

Funções em C++

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
float divisao(int dividendo, int divisor)
{ float quociente;

  quociente = 1.0 * dividendo / divisor;
  return quociente;
}
//-------------------------------------------------------
int main()
{ int a=3, b=4;

  cout << a << "/" << b << "="
       << divisao(a,b) << endl;
  cout << b << "/" << a << "="
       << divisao(b,a) << endl;
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Exemplo

- A constante especial π desempenha um importante papel na matemática. Não é surpresa que haja muitos métodos de obter aproximações numéricas de π. Muitas destas aproximações envolvem operações com séries infinitas. Dentre essa séries temos:

$$\pi = 4\sum_{i=0}^{\infty}\frac{(-1)^i}{2i+1}$$

- Para cálculos práticos, as séries infinitas devem terminar após um número finito de termos, penalizando a precisão do resultado
- Preparar uma função para calcular π de acordo com a série acima. A função deve aceitar, como parâmetro, o valor N, indicando o número de termos a serem utilizados nos cálculos

Exemplo

$$\pi = 4\sum_{i=0}^{\infty}\frac{(-1)^i}{2i+1}$$

```
//-------------------------------------------------------
// Calcula o valor de Pi usando serie com N termos
Função Pi(N : inteiro) : real
  declare soma       : real    // variáveis locais
          i,sinal    : inteiro

  soma ← 0.0
  sinal ← 1
  Para i ← 0 até N-1 Faça
    soma ← soma + sinal/(2.0*i+1.0)
    sinal ← -sinal
  Fim Para
  retorne 4*soma
Fim Função
//-------------------------------------------------------
Algoritmo ImprimePi. Testa função Pi.
Inicio
  declare A : inteiro

  Escreva("Numero de termos = ")
  Leia(A)
  Escreva("Valor de Pi = ",Pi(A)," com ",A," termos")
Fim
```


Exemplo em C++

$$\pi = 4\sum_{i=0}^{\infty}\frac{(-1)^i}{2i+1}$$

```
#include <iostream>
using namespace std;
//-------------------------------------------------------
// Calcula o valor de Pi usando serie com N termos
float Pi(int N)
{ float soma=0.0;
  int i,sinal=1;

  for(i=0; i<=N-1; i++)
  { soma = soma + sinal/(2.0*i+1.0);
    sinal = -sinal;
  }
  return 4*soma;
}
//-------------------------------------------------------
// Testa função Pi.
int main()
{ int A;

  cout << "Numero de termos = ";
  cin >> A;
  cout << "Valor de Pi = " << Pi(A)
       << " com " << A << " termos" << endl;
  return 0;
}
```


Procedimentos

- Em algumas situações desejamos especificar uma operação que não é convenientemente determinada como parte de uma expressão
- Nesses casos, utilizamos outra forma de sub-algoritmo: o procedimento
- Embora a função e o procedimento sejam similares, existem duas diferenças importantes:
  - Numa chamada de procedimento, a execução do programa que o chamou é interrompida, passando o controle ao procedimento chamado. Após a execução do procedimento, o controle retorna ao programa que efetuou a chamada no comando **imediatamente subseqüente**. A execução do programa continua a partir desse ponto
  - Não existe retorno de um único valor como no caso da função. Qualquer valor a ser retornado por um procedimento retorna ao programa que efetuou a chamada por meio de seus parâmetros

Procedimento: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor

Instruções definindo o procedimento **divide**

Procedimento: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor
x	
y	

Instruções definindo o procedimento **divide**

Procedimento: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor
x	3
y	

Instruções definindo o procedimento **divide**

Procedimento: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor
x	3
y	2

Instruções definindo o procedimento **divide**

Procedimento: Fluxo de Controle

```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor
x	3
y	2

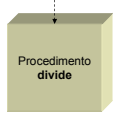
Instruções definindo o procedimento **divide**

Procedimento: Fluxo de Controle

O controle é transferido para o procedimento **divide**

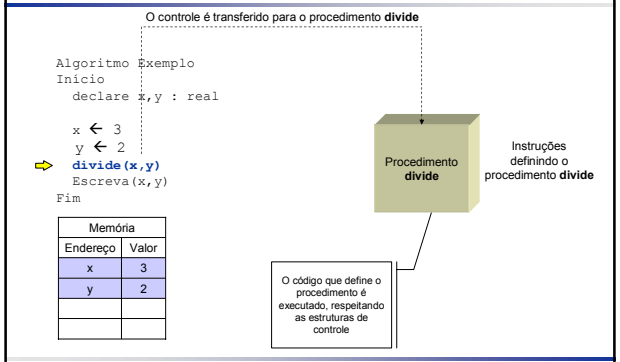
```
Algoritmo Exemplo
Inicio
  declare x,y : real

  x ← 3
  y ← 2
  divide(x,y)
  Escreva(x,y)
Fim
```

Memória	
Endereço	Valor
x	3
y	2

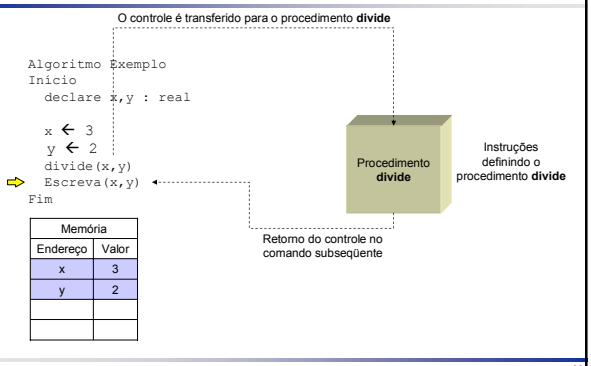


Instruções definindo o procedimento **divide**

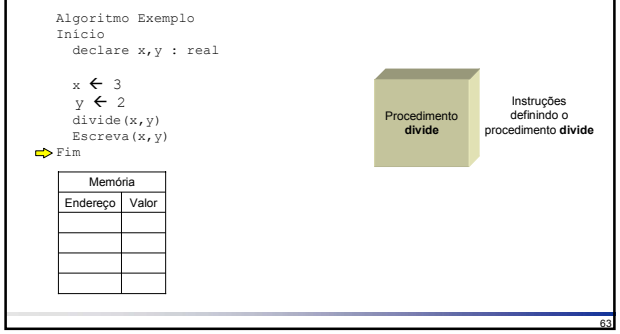
Procedimento: Fluxo de Controle



Procedimento: Fluxo de Controle



Procedimento: Fluxo de Controle



Exemplo

```
//-------------------------------------------------------
// Calcula e imprime quociente e resto entre dois inteiros
Procedimento divide(dividendo, divisor : inteiro)
  declare quociente,resto : inteiro

  quociente ← dividendo / divisor
  resto ← dividendo - quociente * divisor
  Escreva("quociente = ",quociente," resto = ",resto)
Fim Procedimento
//-------------------------------------------------------
Algoritmo TestaDivide
Início
  declare a,b : inteiro

  a ← 5
  b ← 3
  divide(a,b)
  divide(a*b-1,b+1)
Fim
```


Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```


Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica um procedimento. Em C/C++ todo procedimento pode ser visto como uma função que não retorna valor algum

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica o nome do procedimento. Neste caso, **divide**

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica o tipo do primeiro parâmetro. Neste caso, é um inteiro.

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica o nome do primeiro parâmetro. Neste caso, **dividendo**

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica o tipo do segundo parâmetro. Neste caso, é um inteiro.

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Indica o nome do segundo parâmetro. Neste caso, **divisor**

Procedimentos em C++

```
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
```

Declara duas variáveis locais do tipo inteiro.

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```


Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	5
divisor	3

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	5
divisor	3
quociente	
resto	

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	5
divisor	3
quociente	1
resto	

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	5
divisor	3
quociente	1
resto	2

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	5
divisor	3
quociente	1
resto	2

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

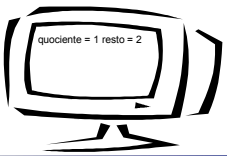
Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

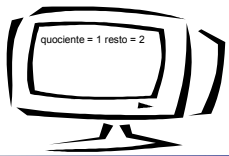


Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	14
divisor	4



Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	14
divisor	4
quociente	
resto	

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	14
divisor	4
quociente	3
resto	

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	14
divisor	4
quociente	3
resto	2

Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor
dividendo	14
divisor	4
quociente	3
resto	2

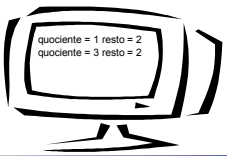
Memória	
Endereço	Valor
a	5
b	3

Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	5
b	3

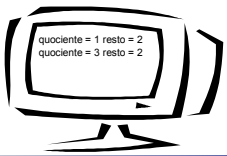


Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
a	5
b	3



Procedimentos em C++

```
#include <iostream>
using namespace std;
//-----------------------------------------
void divide(int dividendo, int divisor)
{ int quociente,resto;

  quociente = dividendo / divisor;
  resto = dividendo - quociente * divisor;
  cout << "quociente = " << quociente
       << " resto = " << resto << endl;
}
//-----------------------------------------
int main()
{ int a=5,b=3;

  divide(a,b);
  divide(a*b-1,b+1);
  return 0;
}
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Passagem de Parâmetros

- Em cada chamada de um procedimento ou função, uma correspondência é estabelecida entre os argumentos (ou parâmetros) da chamada particular e os parâmetros (ou argumentos) de definição do sub-algoritmo
- Existem duas formas de executar essa correspondência:
  - 1) passagem de parâmetros **por valor**
  - 2) passagem de parâmetros **por variável** (ou por referência ou por endereço)

Passagem de Parâmetros

- Passagem por Valor
  - Ao ser efetuada uma chamada de sub-algoritmo, os parâmetros passados por valor são calculados e seus **valores** são atribuídos aos parâmetros de definição; ou seja, os valores são **copiados** para os parâmetros de definição
  - Quaisquer alterações (nos valores das variáveis passadas por valor) efetuadas dentro do sub-algoritmo não causam alterações nos parâmetros de chamada
- Passagem por Variável
  - Na passagem por variável, ao ser efetuada uma chamada de sub-algoritmo, os **endereços** dos parâmetros de chamada são passados aos parâmetros de definição, ou seja, a própria variável de chamada é passada
  - Quaisquer alterações (das variáveis passadas por referência) efetuadas dentro do sub-algoritmo causam a alterações nos parâmetros de chamada

Passagem de Parâmetros

- Em pseudo-código a escolha de passagem por valor ou por variável é efetuada na **definição** de cada parâmetro
- Parâmetros passados por valor são declarados como variáveis comuns
  - `Função f1(a:inteiro, b:real, c:inteiro) : real`
    - parâmetros `a`, `b`, `c` são passados por valor
  - `Procedimento p1(n:inteiro, a[1..100]:real)`
    - parâmetros `n`, `a` são passados por valor
- Parâmetros passados por variável devem ser precedidos pelo símbolo **var** antes do nome do parâmetro
  - `Função f2(var a:inteiro, b:real, var c:real) : real`
    - parâmetros `a` e `c` são passados por variável, parâmetro `b` é passado por valor
  - `Procedimento p2(var n:inteiro, var a[1..100]:real)`
    - parâmetros `n`, `a` são passados por variável

Passagem de Parâmetros C/C++

- Em C/C++ a escolha de passagem por valor ou por variável é efetuada na **definição** de cada parâmetro
- Parâmetros passados por valor são declarados como variáveis comuns
  - `float f1(int a, float b, int c)`
    - parâmetros `a`, `b`, `c` são passados por valor
- Parâmetros passados por referência devem ser precedidos pelo símbolo **&** antes do nome do parâmetro
  - `float f2(int &a, float b, double &c)`
    - parâmetros `a` e `c` são passados por referência, parâmetro `b` é passado por valor
  - `void p2(int &n, float a[])`
    - parâmetros `n`, `a` são passados por variável
- Importante: vetores e matrizes sempre são sempre passados por variável e o símbolo **&** é omitido
  - É desnecessário informar o tamanho dos vetores usados como parâmetros
  - Para matrizes é necessário declarar o tamanho de todas as dimensões exceto a primeira

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //-------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //-------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor
X	0

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor
X	5

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

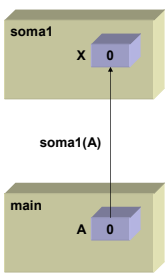
Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```

Linha	A	X
8		
9		
10	0	
11	0	
4	0	0
5	0	5
6	0	
12	0	
13	0	
14		

Passagem por Valor

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma1(int X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma1(A);
12   cout << A << endl;
13   return 0;
14 }
```



Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

& colocado na definição de um parâmetro indica que ele será passado por referência, ou seja, seu endereço será passado

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	0

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor
X	0

Memória	
Endereço	Valor
A	0

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor
X	5

Memória	
Endereço	Valor
A	5

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	5

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	5

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor
A	5

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Memória	
Endereço	Valor

Memória	
Endereço	Valor

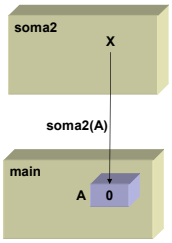
Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```

Linha	A	X
8		
9		
10	0	
11	0	
4	0	0
5	5	5
6	5	
12	5	
13	5	
14		

Passagem por Variável

```
 1 #include <iostream>
 2 using namespace std;
 3 //------------------------
 4 void soma2(int &X)
 5 { X = X + 5;
 6 }
 7 //------------------------
 8 int main()
 9 { int A;
10   A = 0;
11   soma2(A);
12   cout << A << endl;
13   return 0;
14 }
```



Regras de Escopo de Identificadores

- Um identificador consiste em um nome de variável, tipo de dado, sub-algoritmo ou constante simbólica
- O **escopo** de um identificador é a região do programa na qual ele pode ser referenciado
  - Parâmetros e variáveis locais de um sub-algoritmo só podem ser referenciados diretamente dentro do próprio sub-algoritmo; nenhum outro sub-algoritmo pode fazer referência a eles
- Os parâmetros e as variáveis definidas em um sub-algoritmo são **variáveis locais**
  - Uma variável local é criada (alocada na memória) no momento em que o sub-algoritmo que a define é chamado
  - Uma variável local é liberada da memória no momento em que o sub-algoritmo que a define termina
  - O escopo de uma variável local é dentro do sub-algoritmo que a define
- Caso um mesmo identificador (nome de variável) seja declarado em sub-algoritmos distintos, esses identificadores são considerados distintos entre si (variáveis distintas)
- O uso de variáveis locais minimiza a ocorrência de "efeitos colaterais" em programação: o programador pode definir e utilizar as variáveis que desejar em um sub-algoritmo sem interferir com outros sub-algoritmos
- As variáveis definidas na "camada" mais externa de um programa são denominadas **globais** e têm sua existência durante toda a execução do programa
- O uso de variáveis globais deve ser evitado, pois elas podem ser alteradas por quaisquer sub-algoritmos

Exemplo 1

```
#include <iostream>
#include <iomanip>
using namespace std;
//-----------------------------------------------
int i;      // variavel global
//-----------------------------------------------
void escreve(int n)
{ cout << "Imprimindo de 1 ate " << n << endl;
  for(i=1; i<=n; i++)
    cout <<  setw(4) << i;
  cout << endl;
}
//-----------------------------------------------
int main()
{ cout << "Qtde a ser impressa ";
  cin >> i;
  cout << "Qtde = " << i << endl;
  escreve(i);
  cout << "Qtde = " << i << endl;
  return 0;
}
```

```
Qtde a ser impressa 7
Qtde = 7
Imprimindo de 1 ate 7
   1   2   3   4   5   6   7
Qtde = 8
```


Exemplo 2

```
#include <iostream>
#include <iomanip>
using namespace std;
//-----------------------------------------------
void escreve(int n)
{ int i;      // variavel local

  cout << "Imprimindo de 1 ate " << n << endl;
  for(i=1; i<=n; i++)
    cout <<  setw(4) << i;
  cout << endl;
}
//-----------------------------------------------
int main()
{ int i;      // variavel local

  cout << "Qtde a ser impressa ";
  cin >> i;
  cout << "Qtde = " << i << endl;
  escreve(i);
  cout << "Qtde = " << i << endl;
  return 0;
}
```

```
Qtde a ser impressa 7
Qtde = 7
Imprimindo de 1 ate 7
   1   2   3   4   5   6   7
Qtde = 7
```


Exemplo 3

```
#include <iostream>
using namespace std;
//-----------------------------------------------------------
// Calcula a media dos elementos 1 ate n do vetor v
float media(int n, float v[])
{ int i;
  float soma=0.0;

  for(i=1; i<=n; i++)
    soma = soma + v[i];
  return soma / n;
}
//-----------------------------------------------------------
int main()
{ int i;
  float x[]={0,10,20,30,40,50},y[]={0,1,5,10,15,20,25,30,25,40,45};

  cout << "Media x=" << media(5,x) << endl;
  cout << "Media y=" << media(10,y) << endl;
  return 0;
}
```

Lembre-se que todo vetor/matriz em C/C++ é sempre passado por referência e que o símbolo "&" não deve ser utilizado nesse caso. Note também que é desnecessário informar o tamanho dos vetores usados como parâmetros

```
Media x=30.00
Media y=21.60
```


Exemplo 4

```
#include <iostream>
using namespace std;

const int Max=5;
//-----------------------------------------------------------
// Calcula soma dos elementos de uma matriz quadrada ordem n
float soma_matriz(int n, float m[][Max+1])
{ int i, j;
  float soma=0.0;

  for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      soma = soma + m[i][j];
  return soma;
}
//-----------------------------------------------------------
int main()
{ int x, y, n=Max;
  float matriz[Max+1][Max+1];

  for (x=1; x<=n; x++)
    for (y=1; y<=n; y++)
    { cout << "Elemento [" << x << "," << y << "] ? ";
      cin >> matriz[x][y];
    }
  cout << "\nA soma dos elementos da matriz = "
       << soma_matriz(n,matriz) << endl;
  return 0;
}
```

Para matrizes, é necessário declarar o tamanho de todas as dimensões, exceto a primeira dimensão

Exemplo 5

```
#include <iostream>
#include <iomanip>
using namespace std;
//-----------------------------------
// Ordena vetor a[1..N]
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
//-----------------------------------
// Escreve elementos do vetor v[1..n]
void Escrever(int n, float v[])
{ int i;

  for(i=1; i<=n; i++)
    cout << setw(4) << v[i];
  cout << endl;
}
```

```
//-----------------------------------
int main()
{ float x[] = {0,10,30,50,0,2},
    y[] = {0,11,45,7,3,0,-1,5,25,36,8};

  cout << "Vetor x" << endl;
  Escrever(5,x);
  Ordenar(5,x);
  Escrever(5,x);
  cout << "Vetor y" << endl;
  Escrever(10,y);
  Ordenar(10,y);
  Escrever(10,y);
  return 0;
}
```

```
Vetor x
  10  30  50   0   2
   0   2  10  30  50
Vetor y
  11  45   7   3   0  -1   5  25  36   8
  -1   0   3   5   7   8  11  25  36  45
```


Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5

	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	
j	
indice_menor	

	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	
j	
indice_menor	
x	

	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	
indice_menor	
x	

i

	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	
indice_menor	1
x	

i

	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	2
indice_menor	1
x	

	i	j				
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	2
indice_menor	1
x	

	i	j				
	0	1	2	3	4	5
a		10	30	50	0	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	3
indice_menor	1
x	

	i		j			
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	3
indice_menor	1
x	

	i		j			
	0	1	2	3	4	5
a		10	30	50	0	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	4
indice_menor	1
x	

	i			j		
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	4
indice_menor	1
x	

	i			j		
	0	1	2	3	4	5
a		10	30	50	0	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	4
indice_menor	4
x	

	i				j	
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	5
indice_menor	4
x	

	i					j
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	6
indice_menor	4
x	

	i					j
	0	1	2	3	4	5
a		10	30	50	0	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	6
indice_menor	4
x	

	i					j
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	6
indice_menor	4
x	10

	i					
	0	1	2	3	4	5
a		10	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	6
indice_menor	4
x	10

	i					
	0	1	2	3	4	5
a		0	30	50	0	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	1
j	6
indice_menor	4
x	10

i

	0	1	2	3	4	5
a		0	30	50	10	2

Ordenado

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	4
x	10

i

	0	1	2	3	4	5
a		0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	2
x	10

i

	0	1	2	3	4	5
a		0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	3
indice_menor	2
x	10

i j

	0	1	2	3	4	5
a		0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	3
indice_menor	2
x	10

i j

	0	1	2	3	4	5
a		0	30	50	10	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	4
indice_menor	2
x	10

i j

	0	1	2	3	4	5
a		0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	4
indice_menor	2
x	10

	i		j		
0	1	2	3	4	5
a	0	30	50	10	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	4
indice_menor	4
x	10

	i		j		
0	1	2	3	4	5
a	0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	5
indice_menor	4
x	10

	i		j		
0	1	2	3	4	5
a	0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	5
indice_menor	4
x	10

	i		j		
0	1	2	3	4	5
a	0	30	50	10	2

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	5
indice_menor	5
x	10

	i		j		
0	1	2	3	4	5
a	0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	5
x	10

	i				j
0	1	2	3	4	5
a	0	30	50	10	2

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	5
x	30

	0	1	2	3	4	5
a		0	30	50	10	2

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	5
x	30

	0	1	2	3	4	5
a		0	2	50	10	2

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	2
j	6
indice_menor	5
x	30

	0	1	2	3	4	5
a		0	2	50	10	30

i

Ordenado

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	5
x	30

	0	1	2	3	4	5
a		0	2	50	10	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	3
x	30

	0	1	2	3	4	5
a		0	2	50	10	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	4
indice_menor	3
x	30

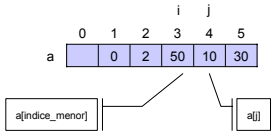
	0	1	2	3	4	5
a		0	2	50	10	30

i j

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

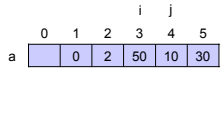
Memória	
Endereço	Valor
N	5
i	3
j	4
indice_menor	3
x	30



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

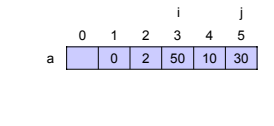
Memória	
Endereço	Valor
N	5
i	3
j	4
indice_menor	4
x	30



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

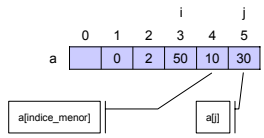
Memória	
Endereço	Valor
N	5
i	3
j	5
indice_menor	4
x	30



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

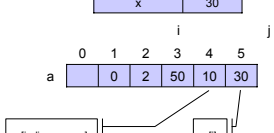
Memória	
Endereço	Valor
N	5
i	3
j	5
indice_menor	4
x	30



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

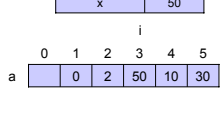
Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	4
x	30



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	4
x	50



Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	10	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	3
j	6
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i

Ordenado

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	5
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i j

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	5
indice_menor	4
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i j

a[indice_menor] a[j]

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	5
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i j

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i j

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	50	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	30	30

i

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	4
j	6
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	30	50

i

Ordenado

Exemplo 5

```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor
N	5
i	5
j	6
indice_menor	5
x	50

	0	1	2	3	4	5
a		0	2	10	30	50

Exemplo 5

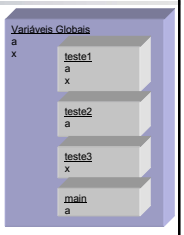
```
void Ordenar(int N, float a[])
{ int i,j,indice_menor;
  float x;

  for(i = 1; i <= N-1; i++)
  { indice_menor = i;
    for(j = i+1; j <= N; j++)
      if(a[indice_menor] > a[j])
        indice_menor = j;
    x = a[i];
    a[i] = a[indice_menor];
    a[indice_menor] = x;
  }
}
```

Memória	
Endereço	Valor

	0	1	2	3	4	5
a		0	2	10	30	50

Exemplo 6

```
#include <iostream>
using namespace std;
int a=3,x=5;
//-------------------------------------------------------------------
void teste1(int x)
{ int a=7;
  cout << "teste1 antes incremento: a=" << a ",x=" << x << endl;
  a++; x++;
  cout << "teste1 depois incremento: a=" << a ",x=" << x << endl;
}
//-------------------------------------------------------------------
void teste2(int a)
{ cout << "teste2 antes incremento: a=" << a ",x=" << x << endl;
  a++; x++;
  cout << "teste2 depois incremento: a=" << a ",x=" << x << endl;
}
//-------------------------------------------------------------------
void teste3(int x)
{ cout << "teste3 antes incremento: a=" << a ",x=" << x << endl;
  a++; x++;
  cout << "teste3 depois incremento: a=" << a ",x=" << x << endl;
}
//-------------------------------------------------------------------
int main()
{ int a=10;
  cout << "principal: a=" << a << ",x=" << x << endl;
  teste1(a);
  cout << "principal: a=" << a << ",x=" << x << endl;
  teste2(a);
  cout << "principal: a=" << a << ",x=" << x << endl;
  teste3(a);
  cout << "principal: a=" << a << ",x=" << x << endl;
  return 0;
}
```




```
principal: a=10,x=5
teste1 antes incremento: a=7,x=10
teste1 depois incremento: a=8,x=11
principal: a=10,x=5
teste2 antes incremento: a=10,x=5
teste2 depois incremento: a=11,x=6
principal: a=10,x=6
teste3 antes incremento: a=3,x=10
teste3 depois incremento: a=4,x=11
principal: a=10,x=6
```


Resumo

- Nesta aula vimos os dois tipos de sub-algoritmos existentes: funções e procedimentos
- A função sempre retorna um valor no ponto onde foi chamada; já o procedimento pode retornar vários valores e seu retorno ao programa que chama é efetuado no comando subseqüente
- Variáveis declaradas dentro de um sub-algoritmo (variáveis locais) têm sua existência somente quando o sub-algoritmo é executado e deixam de existir ao término da execução do sub-algoritmo
- Existem duas formas de correspondência entre parâmetros de definição e de chamada: por valor e por variável