

# BCC 201 - Introdução à Programação I

## Procedimentos e Funções II

Guillermo Cámara-Chávez  
UFOP

# Sub-algoritmos I

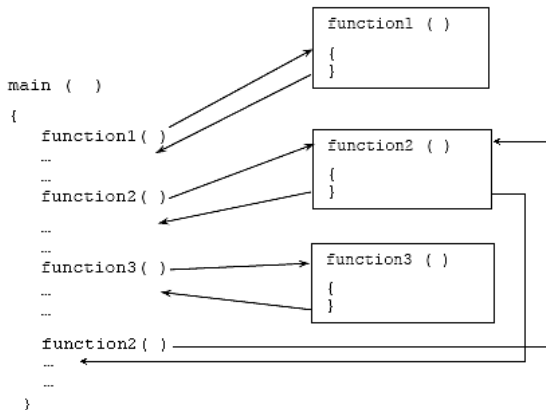
- ▶ Sub-algoritmos são **blocos de instruções** que realizam **tarefas específicas**
- ▶ O código de um sub-algoritmo é **carregado uma vez** e pode ser **executado quantas vezes for necessário**
- ▶ Assim, os **programas** tendem a **ficar menores e mais organizados**, uma vez que o problema pode ser dividido

## Sub-algoritmos II

- ▶ 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
- ▶ Com a chamada, o programa chamador é temporariamente suspenso e o controle é passado para o sub-algoritmo que é executado

# Sub-algoritmos III

- ▶ Ao terminar o sub-algoritmo, o controle retorna para o programa chamador



# Sub-algoritmos IV

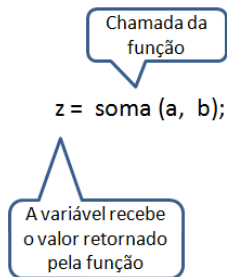
- ▶ Tipos de Sub-algoritmos:
  - ▶ Funções (functions)
  - ▶ Procedimentos (procedures)

# Funções I

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

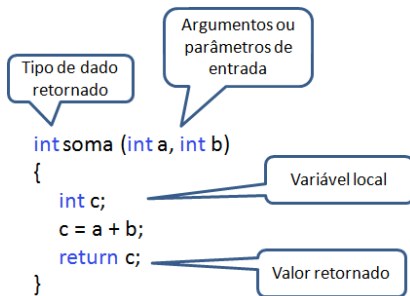
# Funções II

- ▶ Essas funções são utilizadas em expressões como se fossem simplesmente variáveis comuns
- ▶ Como variáveis comuns, as funções têm (ou retornam) um único valor



## Funções III

- ▶ É responsabilidade do programador fornecer o argumento (ou parâmetro) particular necessário para a função efetuar seus cálculos
- ▶ Por exemplo, a função soma tem como parâmetro dois números, retornando um valor também numérico





# Funções IV

- ▶ A utilização de funções afeta o fluxo de controle num programa
- ▶ Quando uma função é chamada, 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 na função

# Funções V

```
scanf("%lf%lf", &a, &b);
```

```
r = sqrt(a) + pow(b, 2);
```

```
printf("r = %lf", r);
```

sqrt(num)

pow(base,exp)

# Funções VI

O controle é transferido para a função sqrt

sqrt(num)

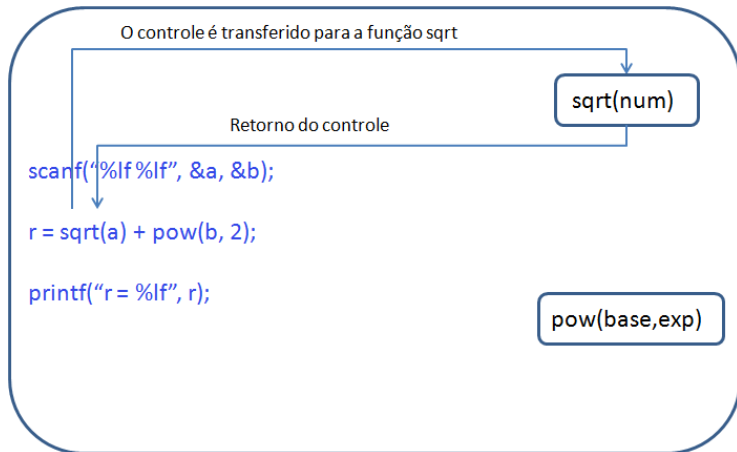
```
scanf("%lf%lf", &a, &b);
```

```
r = sqrt(a) + pow(b, 2);
```

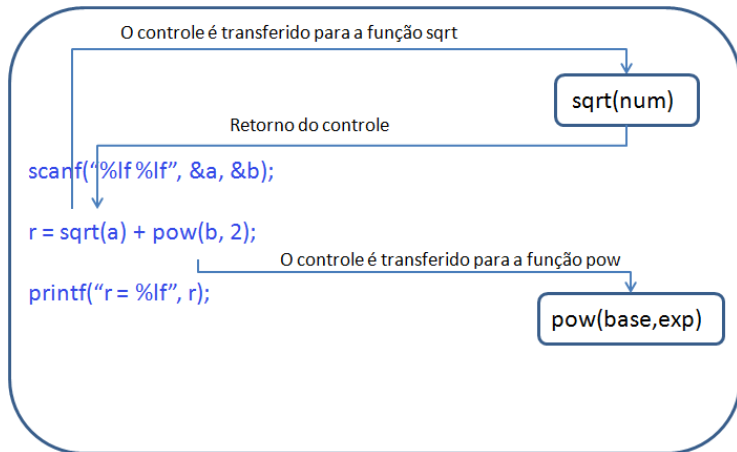
```
printf("r = %lf", r);
```

pow(base,exp)

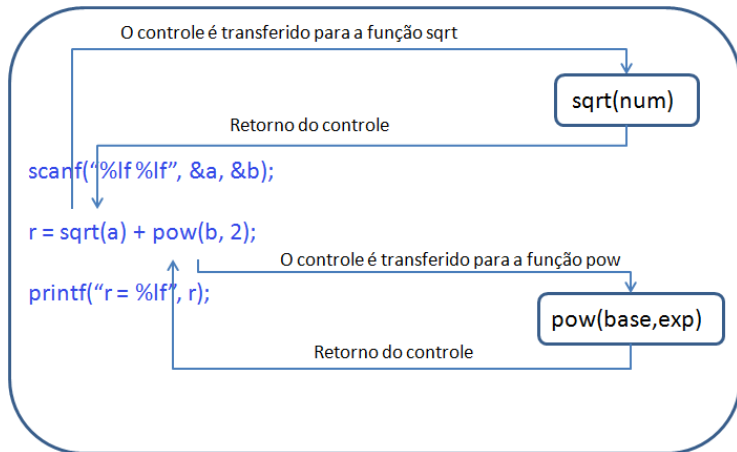
# Funções VII



# Funções VIII



# Funções IX



# Funções X

- ▶ Em algumas situações, o programador gostaria de utilizar (definir) novas funções
- ▶ Por analogia, na Matemática, escreve-se (ou define-se) uma função numa forma geral, 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)
  - ▶  $f(1) = 1^2 - 3(1) + 2 = 0$
  - ▶  $f(-1) = (-1)^2 - 3(-1) + 2 = 6$

# Funções XI

- ▶ Uma vez definida a nova função, ela pode ser utilizada sempre que necessária, mesmo dentro de outras (novas) 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



# Funções XII

- ▶ 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  $g(2, 3)$

# Exercícios I

Encontrar o máximo elemento entre dois números inteiros

## Exercícios II

```
int Maximo(int , int );
int main(){
    int num1, num2;
    printf("Inserir dois numeros");
    scanf("%d %d", &num1, &num2);
    printf("O maior entre %d e %d eh %d \n",
           num1, num2, Maximo(num1, num2));
    return 0;
}
int Maximo(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

## Exercícios III

Calcular o máximo de 3 número, utilizar a função já implementada para encontrar o máximo de dois números

## Exercícios IV

```
int Maximo(int , int);
int Maximo3(int , int , int);
int main(){
    int num1, num2, num3;
    printf("Inserir tres numeros");
    scanf("%d %d %d", &num1, &num2, &num3);
    printf("O maior entre %d, %d e %d eh %d \n",
           num1, num2, num3, Maximo3(num1, num2, num3));
    return 0;
}
int Maximo(int a, int b){
    if (a > b) return a;
    else return b;
}
int Maximo3(int a, int b, int c){
    return Maximo(a, Maximo(b,c));
}
```

# Procedimentos I

- ▶ 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

# Procedimentos II

- ▶ 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 chamador no comando imediatamente subsequente. 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 volta através de seus parâmetros

# Procedimentos III

Divide()

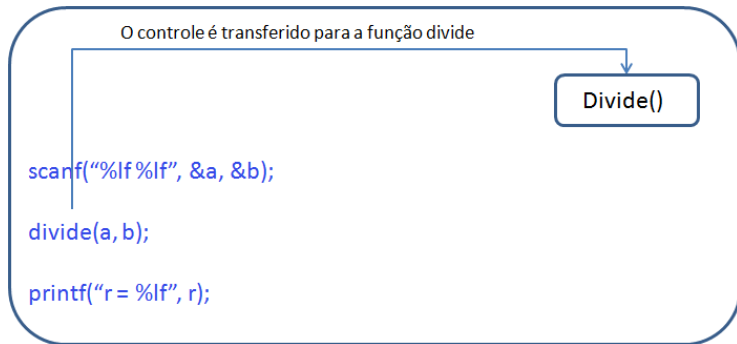
```
scanf("%lf%lf", &a, &b);
```

```
divide(a, b);
```

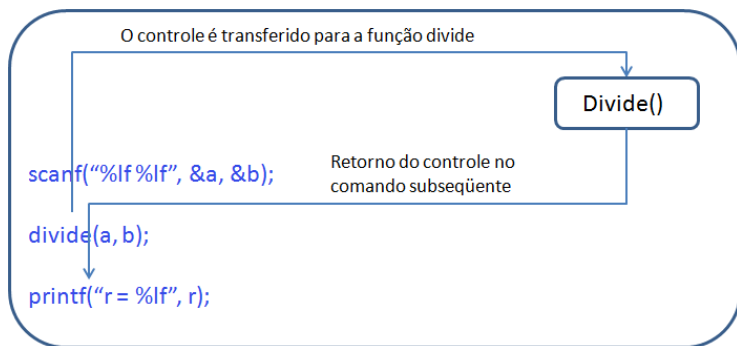
```
printf("r = %lf", r);
```



## Procedimentos IV



# Procedimentos V



## Procedimentos VI

Criar um procedimento que calcule a divisão de dois números.  
Mostrar o resultado dentro do procedimento

# Procedimentos VII

```
void divide(double , double);
int main(){
    double num1, num2;
    printf("Inserir dois números");
    scanf("%lf %lf", &num1, &num2);
    divide(num1, num2);
    return 0;
}
void divide(double a, double b)
{
    if (b != 0)
        printf("Resultado: %lf \n", a/b);
    else
        printf("Nao foi possivel realizar a divisao");
}
```

## Procedimentos VIII

Dados dois números naturais  $m$  e  $n$  e duas seqüências ordenadas com  $m$  e  $n$  números inteiros, obter uma única seqüência ordenada contendo todos os elementos das seqüências originais sem repetição.

Implementar a função que:

1. combina os vetores ordenados

Implementar os procedimentos que:

1. insere dados em um vetor
2. ordena um vetor, e
3. imprime o conteúdo de um vetor

# Procedimentos IX

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
    colocar A[i] em C[k]  
senão  
    Se B[j] < A[i]  
        colocar B[j] em C[k]  
    senão (dois 2 são iguais)  
        colocar B[j] ou A[i] em C[k]  
    fim_se  
fim_se
```

# Procedimentos X

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XI

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1											
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```



## Procedimentos XII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1											
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
    colocar A[i] em C[k]  
senão  
    Se B[j] < A[i]  
        colocar B[j] em C[k]  
    senão (dois 2 são iguais)  
        colocar B[j] ou A[i] em C[k]  
    fim_se  
fim_se
```

# Procedimentos XIII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3										
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XIV

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3										
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XV

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4									
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XVI

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4									
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XVII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5								
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
    colocar A[i] em C[k]  
senão  
    Se B[j] < A[i]  
        colocar B[j] em C[k]  
    senão (dois 2 são iguais)  
        colocar B[j] ou A[i] em C[k]  
    fim_se  
fim_se
```

# Procedimentos XVIII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5								
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XIX

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8							
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

Se  $A[i] < B[j]$  ?

colocar  $A[i]$  em  $C[k]$

senão

Se  $B[j] < A[i]$

colocar  $B[j]$  em  $C[k]$

senão (dois 2 são iguais)

colocar  $B[j]$  ou  $A[i]$  em  $C[k]$

fim\_se

fim\_se



# Procedimentos XX

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8							
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
    colocar A[i] em C[k]  
senão  
    Se B[j] < A[i]  
        colocar B[j] em C[k]  
    senão (dois 2 são iguais)  
        colocar B[j] ou A[i] em C[k]  
    fim_se  
fim_se
```

# Procedimentos XXI

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10						
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XXII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10						
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XXIII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20					
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

## Procedimentos XXIV

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20					
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

Se  $A[i] < B[j]$  ?

colocar  $A[i]$  em  $C[k]$

senão

Se  $B[j] < A[i]$

colocar  $B[j]$  em  $C[k]$

senão (dois 2 são iguais)

colocar  $B[j]$  ou  $A[i]$  em  $C[k]$

fim\_se

fim\_se

# Procedimentos XXV

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20	21				
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

Se  $A[i] < B[j]$  ?

colocar  $A[i]$  em  $C[k]$

senão

Se  $B[j] < A[i]$

colocar  $B[j]$  em  $C[k]$

senão (dois 2 são iguais)

colocar  $B[j]$  ou  $A[i]$  em  $C[k]$

fim\_se

fim\_se

# Procedimentos XXVI

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20	21				
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

```
Se A[i] < B[j] ?  
  colocar A[i] em C[k]  
senão  
  Se B[j] < A[i]  
    colocar B[j] em C[k]  
  senão (dois 2 são iguais)  
    colocar B[j] ou A[i] em C[k]  
  fim_se  
fim_se
```

# Procedimentos XXVII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20	21				
0	1	2	3	4	5	6	7	8	9	10	11

Se terminou a leitura de algum  
vetor?

Se terminou A

passar os elementos de B  
para C

senão

passar os elementos de A  
para C

fim\_se

fim\_se

↑  
k



# Procedimentos XXVIII

A

3	4	8	10	20	21
0	1	2	3	4	5

↑  
i

B

1	5	8	20	30	40
0	1	2	3	4	5

↑  
j

C

1	3	4	5	8	10	20	21	30	40		
0	1	2	3	4	5	6	7	8	9	10	11

↑  
k

Se terminou a leitura de algum  
vetor?

Se terminou A

passar os elementos de B  
para C

senão

passar os elementos de A  
para C

fim\_se

fim\_se

# Procedimentos XXIX

```
#define N 5
void Ordena(int vet[], int n);
void Insere(int vet[], int n);
int Merge(int vet1[], int vet2[], int vet3[],
          int n1, int n2);
void Print(int vet[], int n);

int main()
{
    int A[N], B[N], C[2*N], nC = 0;
    printf("\n Dados Vetor 1");
    Insere(A, N); Ordena(A, N); Print(A, N);
    printf("\n Dados Vetor 2 \n");
    Insere(B, N); Ordena(B, N); Print(B, N);
    nC = Merge(A, B, C, N, N);
    printf("\n Mostrando vetor 3");
    Print(C, nC);
    return 0;
}
```

# Procedimentos XXX

```
void Ordena(int vet[], int n)
{
    int i, j, tmp;
    for (i = 1; i < n; i++)
        for (j = n-1; j >= i; j--)
            if (vet[j] < vet[j-1])
            {
                tmp = vet[j];
                vet[j] = vet[j-1];
                vet[j-1] = tmp;
            }
}
```

# Procedimentos XXXI

```
void Insere(int vet[], int n)
{
    int i;
    printf("\n Insere %d numero", n);
    for (i = 0; i < n; i++)
    {
        printf("\n vet[%d] = ", i);
        scanf("%d", &vet[i]);
    }
}
```

## Procedimentos XXXII

```
void Print(int vet[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("\n vet[%d] = %d", i, vet[i]);
}
```

## Procedimentos XXXIII

```
int Merge(int vet1[], int vet2[], int vet3[],
          int n1, int n2)
{
    int i, j, k;
    i = j = k = 0;
    while(i < n1 && j < n2)
    {
        if (vet1[i] < vet2[j])
            vet3[k++] = vet1[i++];
        else
        {
            if (vet1[i] == vet2[j])
                i++;
            vet3[k++] = vet2[j++];
        }
    }
    ...
}
```

## Procedimentos XXXIV

```
int Merge(int vet1 [], int vet2 [], int vet3 [],  
          int n1, int n2)  
{  
    ...  
    if (i > j)  
        for (; j < n2; j++)  
            vet3[k++] = vet2[j];  
    else  
        for (; i < n1; i++)  
            vet3[k++] = vet1[i];  
  
    return k;  
}
```

FIM