

# Programação 1

## Aula 5

Valeri Skliarov, Prof. Catedrático

Email: [skl@ua.pt](mailto:skl@ua.pt)

URL: <http://sweet.ua.pt/skl/>

Departamento de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

<http://elearning.ua.pt/>

# Revisão da aula anterior

# Funções

```
public static int max(int i1, int i2)
{ return i1 > i2 ? i1 : i2; }
```

```
public static double max(double i1, double i2)
{ return i1 > i2 ? i1 : i2; }
```

```
public static int max(int i1, int i2, int i3)
{ if (i1 > i2 && i1 > i3) return i1;
  else if (i2 > i1 && i2 > i3) return i2;
  return i3;
}
```

```
public static char max(int i1, char c)
{ return i1 > (int)c ? (char)i1 : c; }
```

```
max(3.100000, 5.200000) = 5.200000
max(3, 2) = 3
max(3, 2, 6) = 6
max(48, A) = A
```

```
System.out.printf("max(%f,%f) = %f\n", 3.1, 5.2, max(3.1, 5.2));
System.out.printf("max(%d,%d) = %d\n", 3, 2, max(3, 2));
System.out.printf("max(%d,%d,%d) = %d\n", 3, 2, 6, max(3, 2, 6));
System.out.printf("max(%d,%c) = %c\n", 48, 'A', max(48, 'A'));
```

# Funções

```
public class sobrecarga_de_nomess
{
    static int a=1, b=2;
    public static int max()
    { return a > b ? a : b; }
    public static double max(double i1, double i2)
    { return i1 > i2 ? i1 : i2; }
    public static int max(int i1, int i2, int i3)
    { if (i1 > i2 && i1 > i3) return i1;
      else if (i2 > i1 && i2 > i3) return i2;
      return i3;
    }
    public static char max(int i1, char c)
    { return i1 > (int)c ? (char)i1 : c; }
    public static void main(String[] args)
    {
        System.out.printf("max(%f,%f) = %f\n", 3.1, 5.2, max(3.1, 5.2));
        System.out.printf("max(%d,%d) = %d\n", a, b, max());
        System.out.printf("max(%d,%d,%d) = %d\n", 3, 2, 6, max(3, 2, 6));
        System.out.printf("max(%d,%c) = %c\n", 48, 'A', max(48, 'A'));
    }
}
```

Mal

```
max(3.100000, 5.200000) = 5.200000
max(1, 2) = 2
max(3, 2, 6) = 6
max(48, A) = A
```

# Funções (erros possíveis)

```
public class sobrecarga_de_nomes_erros
{
    static int a=1, b=2;
    static double d=3.3, e=2.2;
    public static int max()
    { return a > b ? a : b; }
    public static double max()
    { return d > e ? d : e; }
    public static void main(String[] args)
    {
        System.out.printf("max(%f,%f) = %f\n", 3.1, 5.2, max());
        System.out.printf("max(%d,%d) = %d\n", a, b, max());
    }
}
```

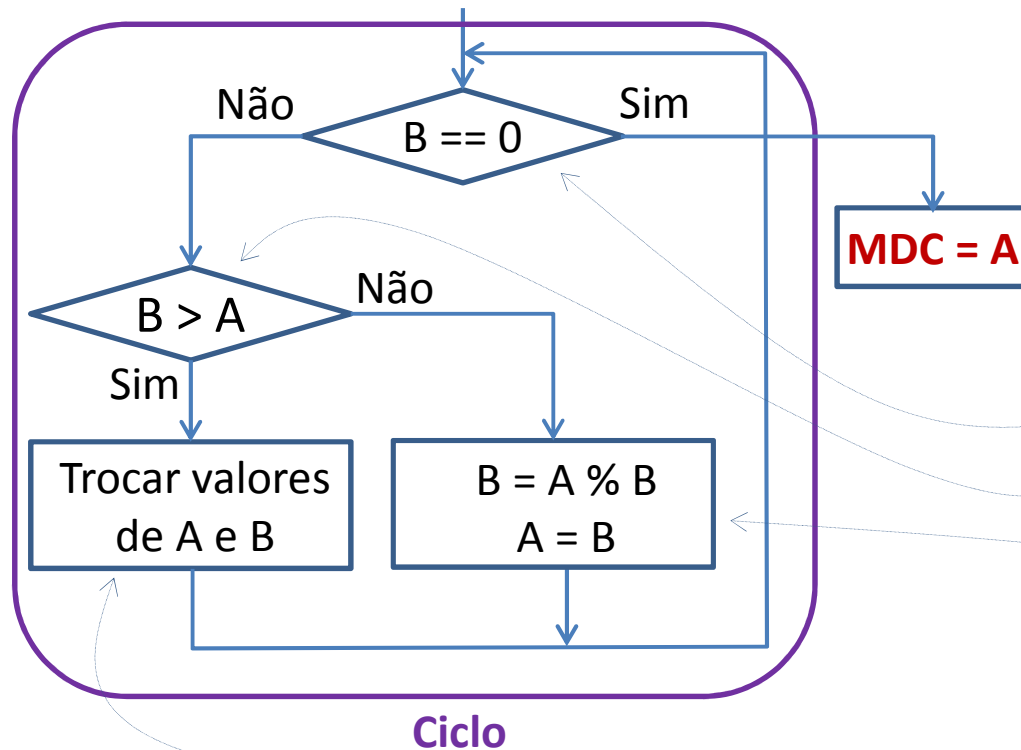
Mal

Erro!!!

Duplicate method max()

# Funções

**Exemplo:** Escreva uma função que recebe dois números inteiros e devolve o seu divisor máximo comum (MDC) através do algoritmo de Euclides.



```
int tmp;
int A,B;
System.out.print("Introduza A: ");
A = sc.nextInt();
System.out.print("Introduza B: ");
B = sc.nextInt();
while (B>0)
{ if (B > A) { tmp=A; A=B; B=tmp;}
  else      { tmp=B; B=A%B; A=tmp;}
}
System.out.println("MDC = "+A);
```

# Funções

Escreva uma função que recebe dois números inteiros e devolve o seu divisor máximo comum (MDC) através do algoritmo de Euclides.

```
import java.util.*;
public class DMC {
    public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        int A,B;
        System.out.print("Introduza A: ");
        A = sc.nextInt();
        System.out.print("Introduza B: ");
        B = sc.nextInt();
        System.out.println("MDC = "+div(A,B));
    }
    public static int div(int A, int B) {
        int tmp;
        while (B>0)
        { if (B > A) { tmp=A; A=B; B=tmp;}
          else      { tmp=B; B=A%B; A=tmp;}
        }
        return A;
    }
}
```

# Funções

**Exemplo:** Escreva uma função que recebe oito números inteiros e devolve o seu divisor máximo comum (MDC) através do algoritmo de Euclides.

```
import java.util.*;
public class DMC {
    public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        int A=3303375, B=20809539, C=127666539, D=19533,
            E=1147851, F=1320201, G=20980740, H=688479651;
        System.out.println("MDC = "+div(div(A,B),div(C,D)), div(div(E,F),div(G,H)));
    }
    public static int div(int A, int B) {
        int tmp;
        while (B>0)
        { if (B > A) { tmp=A; A=B; B=tmp;}
          else      { tmp=B; B=A%B; A=tmp;}
        }
        return A;
    }
}
```

MDC = 1149



# Funções recursivas

Escreva um programa que permite calcular o fatorial de N ( $1 \leq N \leq 10$ ) utilizando uma função recursiva

```
import java.util.*;
public class fact_rec
{ static Scanner sc = new Scanner(System.in);
  public static int lerPositivo()  {
    int x;
    do {
      System.out.print("Valor positivo: ");
      x = sc.nextInt();
    } while(x < 0);
    return x;
  }
  public static void main(String[] args)  {
    int N = lerPositivo();
    while(N > 10 || N < 1) {
      System.out.println("o número deve ser <= 10 e >= 1");
      N = lerPositivo();
    }
    System.out.printf("fatorial de %d = %d\n", N, fact(N) );

    // .....
  }
```

```
public static int fact(int N)  {
  int fatorial = 1;
  for (int i = 1; i <= N; i++)
    fatorial *= i;
  return fatorial;
}
```

```
public static int fact(int N)  {
  int fatorial = N;
  if (fatorial > 1) fatorial *= fact(N-1);
  return fatorial;
}
```

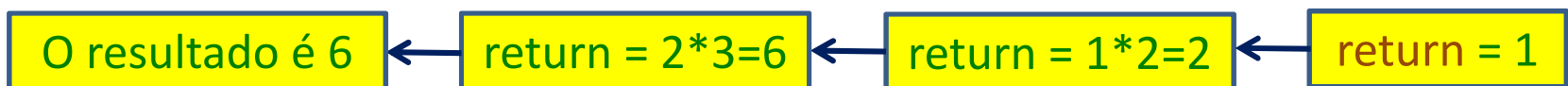
# Funções recursivas

```
public static int fact(int N)    {  
    int fatorial = N;  
    if (fatorial>1) fatorial *= fact(N-1);  
    return fatorial;  
}
```

fact(3)



```
if (fatorial>1) fatorial *= fact(N-1);  
return fatorial;
```



Temos **return** sempre depois da chamada da função **fact**

# Funções recursivas

```
public static int fact(int N)    {  
    int fatorial = N;  
    if (fatorial>1) fatorial *= fact(N-1);  
    System.out.println("valor intermedio = "+fatorial);  
    return fatorial;            }  
}
```

Não faz parte de if

fact(3)

fact(4)

fact(5)

```
Valor positivo: 3  
valor intermedio = 1  
valor intermedio = 2  
valor intermedio = 6  
fatorial de 3 = 6
```

```
Valor positivo: 4  
valor intermedio = 1  
valor intermedio = 2  
valor intermedio = 6  
valor intermedio = 24  
fatorial de 4 = 24
```

```
Valor positivo: 5  
valor intermedio = 1  
valor intermedio = 2  
valor intermedio = 6  
valor intermedio = 24  
valor intermedio = 120  
fatorial de 5 = 120
```

# Funções recursivas

**Exemplo:** Escreva uma função **recursiva** que recebe dois números inteiros e devolve o seu divisor máximo comum (MDC) através do algoritmo de Euclides.

## Função div iterativa

```
public static int div(int A, int B) {  
    int tmp;  
    while (B>0)  
    { if (B > A) { tmp=A; A=B; B=tmp;}  
      else      { tmp=B; B=A%B; A=tmp;}  
    }  
    return A;  
}
```

## Função div recursiva

```
public static int div(int A, int B) {  
    if (B > A) return div(B,A);  
    else if (B==0) return A;  
    else return div(B,A%B);  
}
```

Introduza A:

164

Introduza B:

246

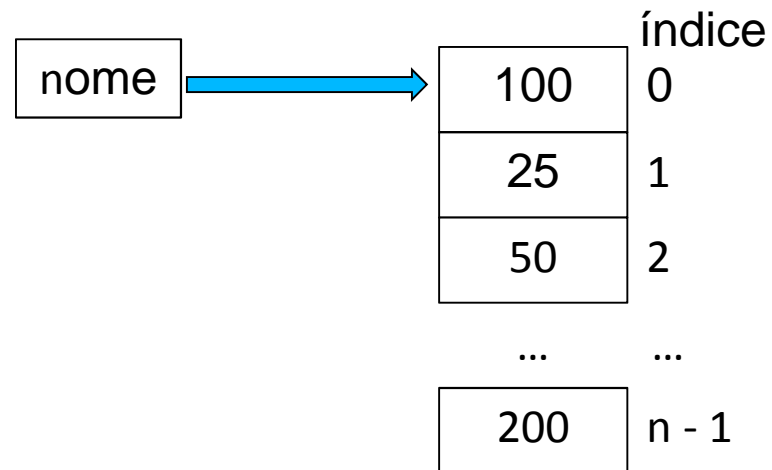
MDC = 82

# Aula 5

- Introdução aos *arrays*
- Declaração de variáveis do tipo *array*
- Acesso aos valores de um *array*
- Arrays como argumentos de funções
- Exemplos

## Arrays (sequências)

- Um *array* é uma organização de memória que se caracteriza pelo fato de ser um agregado de células contínuas, capaz de armazenar um conjunto de valores do mesmo tipo e aos quais se pode aceder de forma indexada.



- Um *array* é identificado pelo nome da variável e o acesso a cada elemento é feito através da respetiva posição.

# Declaração de arrays

- A declaração de um *array* faz-se da seguinte forma:

```
tipo identificador[]; // qualquer tipo...  
identificador = new tipo[dimensão];
```

- Exemplos:

```
double x[];
```

```
x = new double[3]; // array com 3 elementos reais
```

- o *array* x tem os seguintes elementos: x[0], x[1], x[2]

```
int y[];
```

```
y = new int[4]; // array com 4 elementos inteiros
```

- y tem os seguintes elementos: y[0], y[1], y[2], y[3]
- **char** z = **new** char[2]; // array com dois caracteres
- z tem os seguintes elementos: z[0], z[1]

## Acesso aos elementos de um array

- O tipo *array* é concebido como um conjunto de tipos base, sendo apenas possível processar um elemento de cada vez.
- Um elemento do *array* é acedido da forma:

`identificador[índice]`

- Em JAVA, os índices são sempre valores numéricos inteiros positivos sendo o primeiro elemento o zero e o último (dimensão-1).
- O índice pode também ser dado através de uma expressão cujo resultado tem que ser inteiro.
- Caso se tente referenciar um elemento fora do *array* (índice inferior a zero ou superior a (dimensão-1) gera um erro na execução do programa ("acesso fora dos limites").



## Acesso aos elementos de um *array*

- Uma variável do tipo *array* distingue-se de uma variável simples devido ao uso do operador "[ ]" na sua declaração.
- A linguagem JAVA associa a cada *array* um campo de dimensão (`length`) que pode ser usado sempre que seja necessário determinar a sua capacidade de armazenamento. Pode ser usado da forma: `identificador.length`
- É também possível declarar e atribuir um conjunto de valores a um *array* através de uma expressão de inicialização da seguinte forma:
- `int diasDoMes[] = {31, 28, 31, 30, 31, 30, 31, ..., 31};`
- `char letras[] = {'a', 'e', 'i', 'o', 'u'};`
- `// A dimensão é dada pelo número de elementos dentro de {}`

# Leitura e escrita do conteúdo de *arrays*

```
import java.util.*;
public class Array1 {
    static Scanner read = new Scanner(System.in);
    public static void main (String args[]) {
        int i=0, a[] = new int[10];
        while (i < a.length) a[i++] = read.nextInt();
        i=0;
        while (i < a.length) System.out.printf("a[%d] = %d\n",i,a[i++]);
    } }
```

```
33
34
45
12
34
454
232
12
12
434
a[0] = 33
a[1] = 34
a[2] = 45
a[3] = 12
a[4] = 34
a[5] = 454
a[6] = 232
a[7] = 12
a[8] = 12
a[9] = 434
Press any key to continue . . .
```

```
import java.util.*;
public class Array2 {
    static Scanner read = new Scanner(System.in);
    public static void main (String args[]) {
        System.out.print("Quantos elementos ? ");
        int N = read.nextInt(); int a[] = new int[N];
        for(int i=0; i<a.length;i++)
        {
            System.out.print("elemento "+i+" ");
            a[i] = read.nextInt();
        }
        System.out.println("Elementos por ordem inverso:");
        for(int i=a.length-1; i>=0;i--)
            System.out.printf("a[%d] = %d\n",i,a[i]);
    } }
```

```
Quantos elementos ? 5
elemento 0 33
elemento 1 55
elemento 2 77
elemento 3 99
elemento 4 111
Elementos por ordem inverso:
a[4] = 111
a[3] = 99
a[2] = 77
a[1] = 55
a[0] = 33
Press any key to continue . . . _
```

```

import java.util.*;
public class Array1 {
    static Scanner read = new Scanner(System.in);
    public static void main (String args[]) {
        int i=0, a[] = new int[10];
        while (i < a.length) a[i++] = read.nextInt();
        i=0;
        while (i < a.length) System.out.printf("a[%d] = %d\n",i,a[i++]);
    } }

```

Declaração de **array a**  
do tipo **int** que tem  
10 elementos

Tamanho (número  
de elementos) de  
array **a**

Atenção:

String s = "Cuidado"; // s.length() tamanho de String **s**

Atenção:

int a[] = {1,2,3,4,5}; // a.length tamanho de array **a**

Aceso aos  
elementos de array

**a[i++]** significa primeiro utilizar índice **i** e  
depois incrementar índice **i** , por exemplo,  
**i = 3;**  
primeiro **a[3]**  
depois **i = 4**

```

import java.util.*;
public class Array1 {
    static Scanner read = new Scanner(System.in);
    public static void main (String args[])    {
        int i=0, a[] = new int[10];
        while (i < a.length) a[i++] = read.nextInt();
        i=0;
        while (i < a.length) System.out.printf("a[%d] = %d\n",i,a[i++]);
    } }

```

```

33
34
45
12
34
454
232
12
12
434
a[0] = 33
a[1] = 34
a[2] = 45
a[3] = 12
a[4] = 34
a[5] = 454
a[6] = 232
a[7] = 12
a[8] = 12
a[9] = 434
Press any key to continue . . .

```

```

import java.util.*;
public class Array2 {
    static Scanner read = new Scanner(System.in);
    public static void main (String args[]) {
        System.out.print("Quantos elementos ? ");
        int N = read.nextInt(); int a[] = new int[N];
        for(int i=0; i<a.length;i++)
        { System.out.print("elemento "+i+" ");
          a[i] = read.nextInt(); }
        System.out.println("Elementos por ordem inverso:");
        for(int i=a.length-1; i>=0;i--)
            System.out.printf("a[%d] = %d\n",i,a[i]);
    }
}

```

Reserva memória para N  
elementos introduzidos  
através do teclado

Ordem de  
impressão foi  
alterada

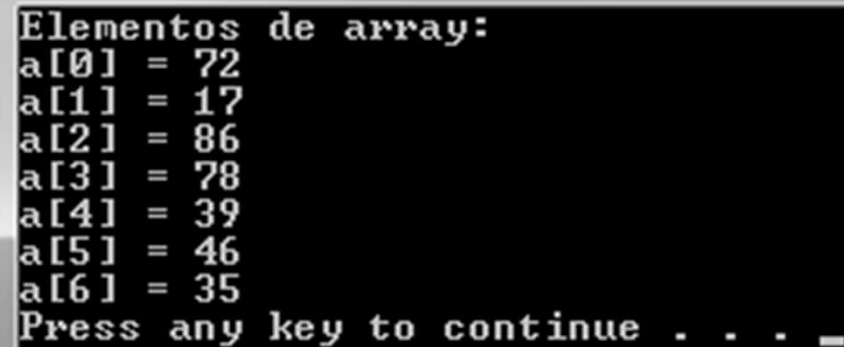
```

Quantos elementos ? 5
elemento 0 33
elemento 1 55
elemento 2 77
elemento 3 99
elemento 4 111
Elementos por ordem inverso:
a[4] = 111
a[3] = 99
a[2] = 77
a[1] = 55
a[0] = 33
Press any key to continue . . . _

```

# Gerar tamanho e elementos de *arrays* aleatoriamente

```
import java.util.*;
public class ArrayAleatorio {
    static Scanner read = new Scanner(System.in);
    static Random rd = new Random();
    public static void main (String args[])    {
        int i=0, a[] = new int[rd.nextInt(10)];
        while(i<a.length) a[i++]=rd.nextInt(100);
        System.out.println("Elementos de array:");
        for(i=0; i<a.length; i++)
            System.out.printf("a[%d] = %d\n",i,a[i]);
    }
}
```



Elementos de array:

a[0]	=	72
a[1]	=	17
a[2]	=	86
a[3]	=	78
a[4]	=	39
a[5]	=	46
a[6]	=	35

Press any key to continue . . . \_

```

import java.util.*;
public class ArrayAleatorio {
    static Scanner read = new Scanner(System.in);
    static Random rd = new Random();
    public static void main (String args[]) {
        int i=0, a[] = new int[rd.nextInt(10)];

        while(i<a.length) a[i++]=rd.nextInt(100);
        System.out.println("Elementos de array:");
        for(i=0; i<a.length; i++)
            System.out.printf("a[%d] = %d\n",i,a[i]);
    }
}

```

1. Incluir a declaração seguinte: **static Random rd = new Random();**

2. Gerar elementos aleatoriamente

3. Atribuir valores de elementos aleatoriamente

```

Elementos de array:
a[0] = 72
a[1] = 17
a[2] = 86
a[3] = 78
a[4] = 39
a[5] = 46
a[6] = 35
Press any key to continue . . . _

```

## Passagem de *arrays* a funções

- Uma variável do tipo *array* é sempre passada por referência a uma função. De fato estamos a passar o endereço do início do *array* em memória.
- Deste modo, uma variável do tipo *array* é sempre um argumento de entrada-saída, podendo o seu conteúdo ser modificado dentro da função.
- Dentro da função podemos saber com quantos elementos foi criado um *array* através do campo `length`.
- No entanto, nem sempre uma sequência está preenchida, pelos que nessas circunstâncias é usual utilizar uma variável inteira adicional, para além do *array*, onde armazenamos o número de elementos preenchidos.



## Exemplo:

```
import java.util.*;
public class array_function {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int lerSequencia(int a[]){
        int n = 0, tmp;
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                           // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return n; // devolvemos o número de valores lidos
    }
    public static void main (String args[])    {
        int a[] = new int[10];
        System.out.printf("%d valores lidos\n",lerSequencia(a));
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
    } }
```

Valor inteiro:	4
Valor inteiro:	7
Valor inteiro:	1
Valor inteiro:	33
Valor inteiro:	118
Valor inteiro:	0

5 valores lidos

4  
7  
1  
33  
118  
>

```

import java.util.*;
public class array_function {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int lerSequencia(int a[]) {
        int n = 0, tmp;
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                               // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return n; // devolvemos o número de valores lidos
    }
    public static void main (String args[]) {
        int a[] = new int[10];
        System.out.printf("%d valores lidos\n",lerSequencia(a));
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
        }
    }
}

```

Função lerSequencia

Argumento **a** da função lerSequencia do tipo array []

Valor de retorno

Memória para array **a** foi reservada fora da função lerSequencia e foi usada dentro da função lerSequencia e dentro da função **main**

Função lerSequencia

```
import java.util.*;
public class array_function {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int lerSequencia(int a[]) {
        int n = 0, tmp;
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                           // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return n; // devolvemos o número de valores lidos
    }
    public static void main (String args[]) {
        int a[] = new int[10];
        System.out.printf("%d valores lidos\n",lerSequencia(a));
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
    }
}
```

Valor inteiro: 4

Valor inteiro: 7

Valor inteiro: 1

Valor inteiro: 33

Valor inteiro: 118

Valor inteiro: 0

5 valores lidos

4

7

1

33

118

>

## Arrays como valor de retorno de uma função

- O valor de retorno de uma função pode ser de qualquer tipo de dados (primitivo ou *referência*).
- Supondo que queríamos copiar o conteúdo de um *array* para outro, podíamos implementar a seguinte função:

```
public static int[] copiaArrays(int[] a, int n){  
    int tmp[] = new int[n];  
    for(int i = 0 ; i < n ; i++){  
        tmp[i] = a[i];  
    }  
    return tmp;  
}
```

- A outra alternativa seria ter uma função que recebia como argumento dois *arrays* já criados.

## Exemplo:

```
import java.util.*;
public class array_function1 {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] lerSequencia(int a[]){
        int n = 0, tmp;  a = new int[10];
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                               // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return a; // devolvemos referência a para array
    }
    public static void main (String args[])    {
        int a[]=null;
        a=lerSequencia(a);
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
    } }
```

Valor inteiro:

Valor inteiro:

Valor inteiro:

Valor inteiro:

Valor inteiro:

Valor inteiro:

55

44

66

11

77

```

import java.util.*;
public class array_function1 {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] lerSequencia(int a[]){
        int n = 0, tmp; a = new int[10];
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                               // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return a; // devolvemos referência a para array
    }
    public static void main (String args[]) {
        int a[]=null;
        a=lerSequencia(a);
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
    }
}

```

Memória para array  
(inicialmente sem elementos)  
foi reservada dentro da função  
**lerSequencia**

Memória já foi reservada  
e podemos utilizar array

Devolvemos referência para  
array com memória reservada  
dentro da função **lerSequencia**

Declaração de array nulo  
(i.e. sem elementos)

Array **a** (nulo) foi passado  
como argumento da função  
**lerSequencia** e foi usado  
como valor de retorno da  
função **lerSequencia**

Memória foi reservada dentro da função  
**lerSequencia** e usada dentro da função **main**

```

import java.util.*;
public class array_function1 {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] lerSequencia(int a[]){
        int n = 0, tmp; a = new int[10];
        do{
            System.out.print("Valor inteiro: ");
            tmp = read.nextInt();
            if(tmp != 0)
                a[n++] = tmp; // armazenamos o valor na posição n
                               // e "avancamos" para a próxima posição
        }while(tmp != 0 && n < a.length);
        return a; // devolvemos referência a para array
    }
    public static void main (String args[])    {
        int a[]=null;
        a=lerSequencia(a);
        for(int i=0; i<a.length ;i++)
            if(a[i]==0) continue;
            else System.out.println(a[i]);
    }
}

```

Memória foi reservada dentro da função lerSequencia e usada dentro da função main

Valor inteiro:	55
Valor inteiro:	44
Valor inteiro:	66
Valor inteiro:	11
Valor inteiro:	77
Valor inteiro:	0

55  
44  
66  
11  
77

## ***Exemplo: soma de elementos de dois arrays com tamanhos iguais***

```
import java.util.*;
public class array_soma {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] soma(int a[], int b[]){
        if(a.length != b.length) return null;
        int c[] = new int[a.length];
        for(int i=0; i < a.length; i++)
            c[i] = a[i] + b[i];
        return c;
    }
    public static void main (String args[])    {
        int a[]={1,2,3,4,5};
        int b[]={10,20,30,40,50};
        int y[] = soma(a,b);
        for(int i=0; i < y.length; i++)
            System.out.printf("%d + %d = %d\n",
                               a[i], b[i], y[i]);
    } }
```

1 + 10 = 11  
2 + 20 = 22  
3 + 30 = 33  
4 + 40 = 44  
5 + 50 = 55



```

import java.util.*;
public class array_soma {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] soma(int a[], int b[]){

        if(a.length != b.length) return null;

        int c[] = new int[a.length];

        for(int i=0; i < a.length; i++)
            c[i] = a[i] + b[i];
        return c;
    }

    public static void main (String args[]) {
        int a[] = {1,2,3,4,5};
        int b[] = {10,20,30,40,50};
        int y[] = soma(a,b);
        for(int i=0; i < y.length; i++)
            System.out.printf("%d + %d = %d\n",
                               a[i], b[i], y[i]);
    }
}

```

Argumentos **a** e **b** da função **soma** do tipo array []

Função soma

Vamos abordar só arrays com tamanhos iguais

Declaração de array para somas e reserva da memória

Cálculos para somas

Devolver referência **c** do array com somas

Arrays **a** e **b** com argumentos

Impressão de resultados

Agora **y** contém somas

```

import java.util.*;
public class array_soma {
    static Scanner read = new Scanner(System.in);
    // Leitura de valores até aparecer o zero
    public static int[] soma(int a[], int b[]){
        if(a.length != b.length) return null;

        int c[] = new int[a.length];

        for(int i=0; i < a.length; i++)
            c[i] = a[i] + b[i];
        return c;
    }

    public static void main (String args[]) {
        int a[] = {1,2,3,4,5};
        int b[] = {10,20,30,40,50};
        int y[] = soma(a,b);
        for(int i=0; i < y.length; i++)
            System.out.printf("%d + %d = %d\n",
                               a[i], b[i], y[i]);
    }
}

```

Função soma

Agora **y** contém somas

$1 + 10 = 11$   
 $2 + 20 = 22$   
 $3 + 30 = 33$   
 $4 + 40 = 44$   
 $5 + 50 = 55$

## Exemplo: calcular média

```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
    public static void main (String args[])    {
        int b[]={10,20,30,40,50};
        System.out.println(calcMedia1(b, b.length));
        System.out.println(calcMedia2(b, b.length));
    }
}
```

30.0  
30.0

```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
    public static void main (String args[])    {
        int b[]={10,20,30,40,50};
        System.out.println(calcMedia1(b, b.length));
        System.out.println(calcMedia2(b, b.length));
    }
}
```

Argumento array

Argumento **n** que é tamanho de *array*.  
É melhor utilizar valor **a.length** dentro da função com só um argumento **a**

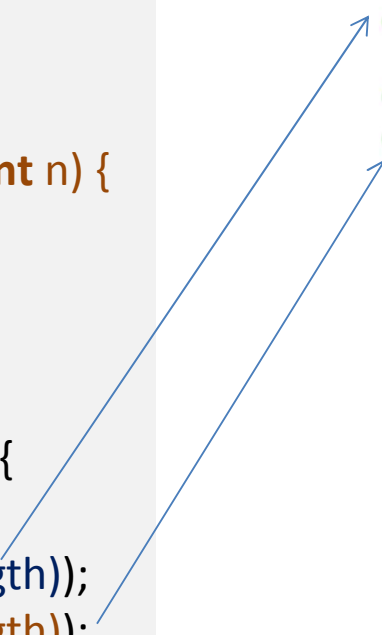
Estas funções são semelhantes

Chamada da primeira função

Chamada da segunda função

```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
    public static void main (String args[])    {
        int b[]={10,20,30,40,50};
        System.out.println(calcMedia1(b, b.length));
        System.out.println(calcMedia2(b, b.length));
    }
}
```

30.0  
30.0



## Exemplo: tempo de execução

```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
    public static void main (String args[])    {
        int b[]={10,20,30,40,50};
        long time,time_end;
        time=System.nanoTime(); System.out.println(calcMedia1(b, b.length));
        time_end=System.nanoTime();
        System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
        time=System.nanoTime(); System.out.println(calcMedia2(b, b.length));
        time_end=System.nanoTime();
        System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
    }
}
```

30.0

measured time (in ms): 1.689

30.0

measured time (in ms): 0.673

Saber o tempo de execução é importante para avaliar e comparar vários algoritmos alternativos

```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
    public static void main (String args[])    {
        int b[]={10,20,30,40,50};
        long time,time_end;
        time=System.nanoTime();
        System.out.println(calcMedia1(b, b.length));
        time_end=System.nanoTime();
        System.out.printf("measured time (in ms): %.3f\n", (double)(time_end-time)/1000000.);
        time=System.nanoTime();
        System.out.println(calcMedia2(b, b.length));
        time_end=System.nanoTime();
        System.out.printf("measured time (in ms): %.3f\n", (double)(time_end-time)/1000000.);
    } }
```

Ler tempo inicial

Ler tempo final

Imprimir a diferença

## *Pode encontrar função **nanoTime()** em classe **System**:*

<https://docs.oracle.com/javase/8/docs/api/java/lang/System.html>

static **String**

**mapLibraryName(String libname)**

Maps a library name into a platform-specific string representing a native library.

static long

**nanoTime()**

Returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.

static void

**runFinalization()**

Runs the finalization methods of any objects pending finalization.

static void

**runFinalizersOnExit(boolean value)**

**Deprecated.**

This method is inherently unsafe. It may result in finalizers being called on live objects while other threads are concurrently manipulating those objects, resulting in erratic behavior or deadlock.

static void

**setErr(PrintStream err)**

Reassigns the "standard" error output stream.



```
import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
}
```

Tempo de execução da primeira função calcMedia1

Tempo de execução da segunda função calcMedia2

```
public static void main (String args[])    {
    int b[]={10,20,30,40,50};
    long time,time_end;
```

```
    time=System.nanoTime();
    System.out.println(calcMedia1(b, b.length));
    time_end=System.nanoTime();
```

```
    System.out.printf("measured time (in ms): %.3f\n", (double)(time_end-time)/1000000.);
```

```
    time=System.nanoTime();
    System.out.println(calcMedia2(b, b.length));
    time_end=System.nanoTime();
```

```
    System.out.printf("measured time (in ms): %.3f\n", (double)(time_end-time)/1000000.);
```

```
    } }
```

Converter **long** para **double**

Dividir por 1000000. para obter tempo em milisegundos

```

import java.util.*;
public class array_soma1 {
    static Scanner read = new Scanner(System.in);
    public static double calcMedia1(int a[], int n){
        int soma = 0;
        double m;
        for(int i = 0 ; i < n ; i++){
            soma += a[i];
        }
        m = (double)soma / n;
        return m;
    }
    public static double calcMedia2(int a[], int n) {
        int soma = 0;
        for(int i = 0; i < n ;soma += a[i++]);
        return (double)soma / n;
    }
}

```

```

public static void main (String args[])    {
    int b[]={10,20,30,40,50};
    long time,time_end;
    time=System.nanoTime(); System.out.println(calcMedia1(b, b.length));
    time_end=System.nanoTime();
    System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
    time=System.nanoTime(); System.out.println(calcMedia2(b, b.length));
    time_end=System.nanoTime();
    System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
} }

```

30.0

measured time (in ms): 1.689

30.0

measured time (in ms): 0.673

# Exemplos com *arrays*

```

public class for_and_array
{
    public static void main(String[] args)
    {
        int a[] = { 1, 2, 3, 4, 5 };
        // pode também declarar array como: int a[] = { 1, 2, 3, 4, 5 };
        for(int i = 0; i < a.length; i++)
            System.out.println("a[" + i + "] = " + a[i]);
        System.out.println("-----");
        for(int i = a.length-1; i >= 0; i--)
            System.out.println("a[" + i + "] = " + a[i]);
    }
}

```

```

a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5

```

```

a[4] = 5
a[3] = 4
a[2] = 3
a[1] = 2
a[0] = 1

```

Leitura de array **a** e impressão de array **a** por ordem inversa

## Exemplo 1:

O seguinte exemplo permite gerar dados aleatoriamente e ordenar dados utilizando uma rede de ordenação:

```
import java.util.*;
public class sorting_network
{
    static Random rand = new Random();
    public static void main(String[] args)
    {
        int N = 64, tmp; // N pode ser qualquer valor de 2**R, R = 0,1,2,3,4,5,6,7,8,9,10,...
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++)
        {
            a[i] = rand.nextInt(1000);
            System.out.println("a[" + i + "] = " + a[i]);
        }
        System.out.println("-----");
        for(int k = 0; k < N/2; k++)
        {
            for(int i = 0; i < a.length/2; i++)
            {
                if (a[2*i] < a[2*i+1]) { tmp = a[2*i]; a[2*i] = a[2*i+1]; a[2*i+1] = tmp; }
            }
            for(int i = 0; i < a.length/2-1; i++)
            {
                if (a[2*i+1] < a[2*i+2]) { tmp = a[2*i+1]; a[2*i+1] = a[2*i+2]; a[2*i+2] = tmp; }
            }
        }
        for(int i = 0; i < a.length; i++) { System.out.printf("%10d; ", a[i]);
            if (((i+1)%10) == 0) System.out.println();
        }
    }
}
```

**int N = 16, tmp;**

```
a[0] = 787  
a[1] = 87  
a[2] = 885  
a[3] = 671  
a[4] = 442  
a[5] = 678  
a[6] = 84  
a[7] = 241  
a[8] = 749  
a[9] = 441  
a[10] = 47  
a[11] = 495  
a[12] = 736  
a[13] = 174  
a[14] = 360  
a[15] = 10
```

Gerar dados aleatoriamente

```
885; 787; 749; 736; 678; 671; 495; 442; 441; 360;  
241; 174; 87; 84; 47; 10; >
```

Dados ordenados

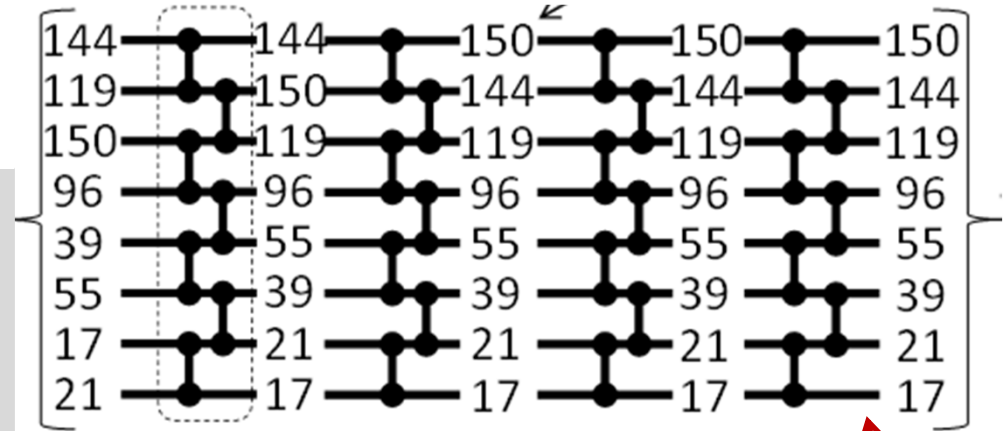
Pode alterar para 4,8,16,32,64,128,256,...

```
import java.util.*;
public class sorting_network
{
    static Random rand = new Random();
    public static void main(String[] args)
    {
        int N = 64, tmp; // N pode ser qualquer valor de 2**R, R = 0,1,2,3,4,5,6,7,8,9,10,...
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++)
        {
            a[i] = rand.nextInt(1000);
            System.out.println("a[" + i + "] = " + a[i]);
        }
        System.out.println("-----");
        for(int k = 0; k < N/2; k++)
        {
            for(int i = 0; i < a.length/2; i++)
            {
                if (a[2*i] < a[2*i+1]) { tmp = a[2*i]; a[2*i] = a[2*i+1]; a[2*i+1] = tmp; }
                for(int i = 0; i < a.length/2-1; i++)
                {
                    if (a[2*i+1] < a[2*i+2]) { tmp = a[2*i+1]; a[2*i+1] = a[2*i+2]; a[2*i+2] = tmp; }
                }
            }
        }
        for(int i = 0; i < a.length; i++) { System.out.printf("%10d", a[i]);
            if (((i+1)%10) == 0) System.out.println();
        }
    }
}
```

Gerar dados aleatoriamente

Rede de ordenação

Imprimir os resultados



## Vamos encontrar o tempo de execução

```
import java.util.*;
public class sorting_network
{
    static Random rand = new Random();
    public static void main(String[] args)
    {
        int N = 64, tmp; // N pode ser qualquer valor de 2**R, R = 0,1,2,3,4,5,6,7,8,9,10,...
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++)
        {
            a[i] = rand.nextInt(1000);
            System.out.println("a[" + i + "] = " + a[i]);
        }
        System.out.println("-----");
        long time=System.nanoTime();
        for(int k = 0; k < N/2; k++)
        {
            for(int i = 0; i < a.length/2; i++)
            {
                if (a[2*i] < a[2*i+1]) { tmp = a[2*i]; a[2*i] = a[2*i+1]; a[2*i+1] = tmp; }
            }
            for(int i = 0; i < a.length/2-1; i++)
            {
                if (a[2*i+1] < a[2*i+2]) { tmp = a[2*i+1]; a[2*i+1] = a[2*i+2]; a[2*i+2] = tmp; }
            }
        }
        long time_end=System.nanoTime();
        System.out.printf("measured time (in ms): %.3f\n", (double)(time_end-time)/1000000.);
        for(int i = 0; i < a.length; i++) { System.out.printf("%10d; ", a[i]);
            if (((i+1)%10) == 0) System.out.println();
        }
    }
}
```



## *Os resultados para N = 16*

`int N = 16, tmp;`

```
a[0] = 69  
a[1] = 874  
a[2] = 695  
a[3] = 379  
a[4] = 979  
a[5] = 729  
a[6] = 432  
a[7] = 700  
a[8] = 719  
a[9] = 929  
a[10] = 526  
a[11] = 728  
a[12] = 926  
a[13] = 39  
a[14] = 908  
a[15] = 847
```

-----  
`measured time (in ms): 0.016`

```
979;      929;      926;      908;      874;      847;      729;      728;      719;      700;  
695;      526;      432;      379;      69;      39; >
```

## Os resultados para N = 1024

int N = 1024, tmp;

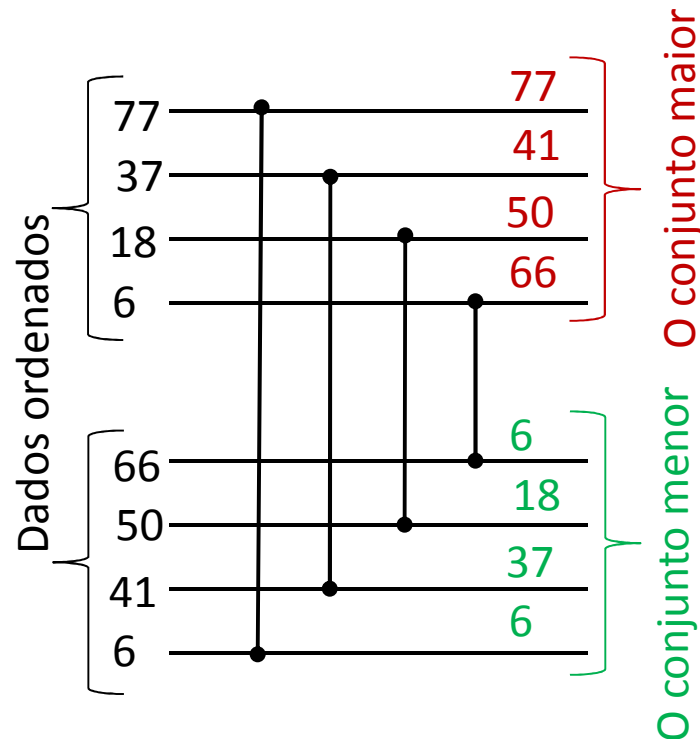
```
a[1020] = 996  
a[1021] = 611  
a[1022] = 401  
a[1023] = 653
```

```
-----  
measured time (in ms): 29.317
```

998;	998;	997;	997;	996;	996;	996;	996;	996;	996;
995;	994;	993;	992;	992;	991;	985;	985;	983;	981;
979;	979;	978;	978;	976;	975;	974;	974;	973;	973;
972;	969;	969;	968;	967;	966;	966;	966;	965;	965;
965;	965;	964;	962;	961;	959;	959;	958;	957;	956;
956;	954;	953;	950;	950;	949;	948;	948;	945;	945;
944;	944;	943;	941;	940;	937;	936;	935;	933;	932;
931;	930;	930;	929;	929;	928;	928;	928;	927;	926;
925;	924;	923;	922;	922;	921;	921;	920;	918;	917;
915;	915;	915;	912;	910;	910;	909;	909;	909;	908;
908;	908;	904;	903;	898;	897;	896;	893;	893;	892;
892;	890;	890;	887;	885;	885;	884;	883;	882;	882;
881;	881;	880;	879;	879;	879;	879;	879;	878;	878;
876;	874;	873;	873;	872;	872;	871;	869;	869;	868;
867;	867;	864;	864;	863;	863;	863;	862;	862;	861;
858;	858;	857;	857;	856;	855;	854;	853;	852;	852;
849;	849;	848;	848;	846;	842;	842;	840;	839;	839;
839;	838;	836;	832;	829;	829;	826;	825;	824;	823;
823;	823;	821;	821;	817;	817;	817;	816;	816;	814;
813;	812;	812;	810;	809;	809;	808;	808;	806;	806;
805;	803;	803;	803;	802;	798;	797;	796;	796;	795;
795;	793;	790;	788;	788;	787;	786;	786;	785;	784;
784;	782;	782;	781;	780;	779;	778;	778;	778;	777;
776;	776;	774;	773;	771;	768;	766;	765;	761;	759;
759;	758;	757;	757;	757;	756;	756;	755;	754;	752;
752;	752;	751;	751;	750;	748;	748;	748;	746;	746;
745;	743;	743;	742;	742;	742;	741;	739;	738;	737;
736;	735;	732;	731;	730;	729;	727;	726;	725;	724;
724;	723;	723;	722;	721;	719;	718;	718;	716;	716;
716;	712;	711;	711;	710;	710;	709;	708;	707;	507;
706;	706;	706;	706;	704;	703;	703;	702;	701;	701;

## Exemplo 2:

Aplicação da rede em baixo a dois conjuntos ordenados permite encontrar o conjunto maior (em cima) e o conjunto menor (em baixo)

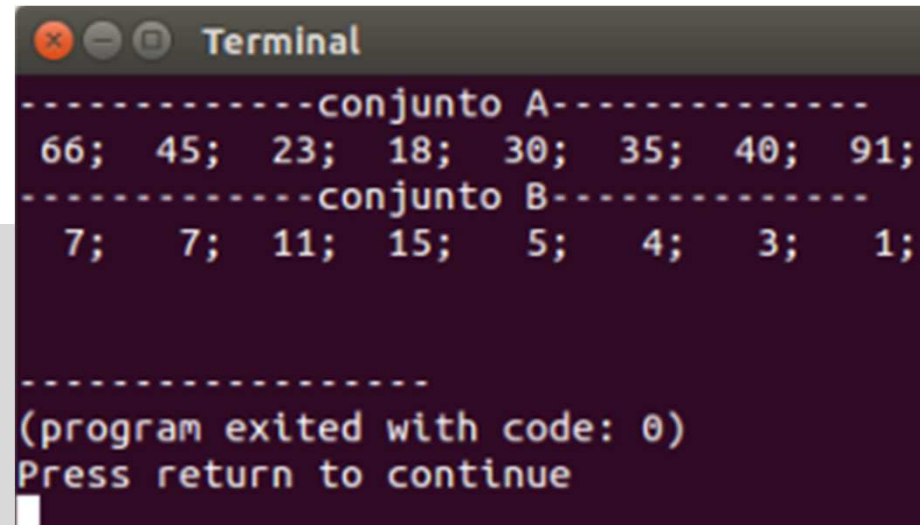


```
public class sort
{
    public static void main(String[] args)
    {
        int[] A = { 66,45,23,18,15,11,7,7 };
        int[] B = { 91,40,35,30,5,4,3,1 };
        int tmp;
        for (int i = 0, j = A.length-1; i < A.length; i++, j--)
            if (A[i] < B[j]) { tmp = A[i]; A[i] = B[j]; B[j] = tmp; }
        System.out.println("-----conjunto A-----");
        for(int i = 0; i < A.length; i++) System.out.printf("%3d", A[i]);
        System.out.println("\n-----conjunto B-----");
        for(int i = 0; i < B.length; i++) System.out.printf("%3d", B[i]);
        System.out.println();
    }
}
```

```

public class sort
{
    public static void main(String[] args)
    {
        int[] A = { 66,45,23,18,15,11,7,7 };
        int[] B = { 91,40,35,30,5,4,3,1 };
        int tmp;
        for (int i = 0, j = A.length-1; i < A.length; i++, j--)
            if (A[i] < B[j]) { tmp = A[i]; A[i] = B[j]; B[j] = tmp; }
        System.out.println("-----conjunto A-----");
        for(int i = 0; i < A.length; i++) System.out.printf("%3d; ",A[i]);
        System.out.println("\n-----conjunto B-----");
        for(int i = 0; i < B.length; i++) System.out.printf("%3d; ",B[i]);
        System.out.println();
    }
}

```



```

Terminal
-----conjunto A-----
66; 45; 23; 18; 30; 35; 40; 91;
-----conjunto B-----
7; 7; 11; 15; 5; 4; 3; 1;

-----
(program exited with code: 0)
Press return to continue

```

```

import java.util.*;
public class funcoes_booleanas
{
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    {
        int a[] = { 0, 1, 2, 3 };
        for(char op = '&';(op == '&') || (op == '|') || (op == '^');)
        { System.out.print("operacao ? ");
          op = sc.next().charAt(0);
          AndOrXor(op,a);
        }
        public static void AndOrXor(char operacao,int a[])
        {
            switch(operacao)
            {
                case '^': System.out.println("operacao ^ :");
                          for (int i = 0; i < a.length; i++)
                              System.out.print(((a[i] & 0x2)>>1) + " " + (a[i] & 0x1) + " " + (((a[i] & 0x2)>>1) ^ (a[i] & 0x1)) + '\n');
                          break;
                case '|': System.out.println("operacao | :");
                          for (int i = 0; i < a.length; i++)
                              System.out.print(((a[i] & 0x2)>>1) + " " + (a[i] & 0x1) + " " + (((a[i] & 0x2)>>1) | (a[i] & 0x1)) + '\n');
                          break;
                case '&': System.out.println("operacao & :");
                          for (int i = 0; i < a.length; i++)
                              System.out.print(((a[i] & 0x2)>>1) + " " + (a[i] & 0x1) + " " + (((a[i] & 0x2)>>1) & (a[i] & 0x1)) +
                              '\n');
                          break;
                default: System.out.println("operacao errada");
            }
        }
    }
}

```

```

Terminal
operacao ? &
operacao & :
0 0 0
0 1 0
1 0 0
1 1 1
operacao ? |
operacao | :
0 0 0
0 1 1
1 0 1
1 1 1
operacao ? ^
operacao ^ :
0 0 0
0 1 1
1 0 1
1 1 0
operacao ? !
operacao errada

-----
(program exited with code: 0)
Press return to continue

```

### Exemplo 3 :

Imprimir a tabela de verdade para operações booleanas ^, |, & aplicadas a 2 variáveis

# Conclusão

Para *arrays* é necessário declarar referências e reservar memória referenciada

Memória pode ser reservada em qualquer função. Pode devolver uma referência para memória reservada e utilizar memória reservada fora da função

Geralmente não pode utilizar *arrays* sem reservar a memória

Passando argumentos do tipo *array* a função pode aceder aos elementos na memória reservada fora da função

A devolução da função *F* duma referência para um *array* permite aceder a memória (reservada dentro da função *F*) fora da função *F*, i.e. noutra função que recebe o valor devolvido



# Erros mais comuns na avaliação erro

(;) ponto e vírgula depois de declaração de funções: `F( ... ); { ... }`

Em muitas outras situações (;) ponto e vírgula foi usado erradamente

Utilização de formatação errada (ex.: %d, %f, etc.)

Uso incorreto de funções

Não deve tentar implementar o código completo a partir de início. É significativamente mais simples implementar e verificar partes, i.e. abordar o projeto complexo parte a parte

Sugestão: verifique cada função autónoma com cuidado antes de integrar esta função no projeto