

# Programas simples em C

**Problema 1.** *Escreve um programa em C que dados dois inteiros indique se são iguais ou qual o maior.*



Utilizar a construção em 5 etapas... quais?



1. Perceber o problema

2. Ideia da resolução

3. Algoritmo

4. Codificação em C

5. Verificação



# Algoritmo

1. Ler um inteiro e guardar em  $n1$
2. Ler um inteiro e guardar em  $n2$
3. Se  $n1 == n2$ , escrever  *$n1$  é igual a  $n2$*
4. Se  $n1 < n2$ , escrever  *$n2$  é maior do que  $n1$*
5. Se  $n1 > n2$ , escrever  *$n1$  é maior do que  $n2$*

ou simplificando:

Ler  $n1, n2$

Se  $n1 == n2$  escrever  *$n1$  é igual a  $n2$*

Se  $n1 < n2$  escrever  *$n2$  é maior que  $n1$*   
senão escrever  *$n1$  é maior do que  $n2$*

# Programa em C

```
#include <stdio.h>
main() {
    int n1,n2;

    printf("Introduz o primeiro inteiro\n");
    scanf("%d", &n1);
    printf("Introduz o segundo inteiro\n");
    scanf("%d",&n2);
    if ( n1 == n2)
        printf("%d igual a %d\n", n1,n2);
    if ( n1 < n2)
        printf("%d maior do que a %d\n", n2,n1);
    else
        printf("%d maior do que a %d\n", n1,n2);
}
```

# Algumas notas sobre o programa em C

- O programa tem uma função: `main`. Esta função tem sempre que existir! ■
- `n1` e `n2` são variáveis que estão declaradas com um *tipo*. Neste caso, têm ambas o tipo `int` (inteiro). ■ As variáveis permitem guardar valores. ■

`n1`    5

`n2`    7 ■

- Existem (e podem definir-se) outros tipos: `float` (racional), `char`, (representa caracteres), etc. ■
- cada instrução termina com um `;` (ponto-e-vírgula) ■
- A mudança de linha não tem significado especial ■

- A indentação das linhas não tem significado mas é normalmente usada para tornar o programa mais legível■
- Uma instrução `if` permite a execução condicional de instruções■

```
if (condicao) instrucao1 [else instrucao2]
```



`instrucao1` é executada se a `condicao` for verdadeira. Se for falsa `instrucao2` é executada se existir.■ Operadores relacionais: `==`, `!=`, `<`, `<=`, `>`, `>=`...

- As chamadas à função `printf()`, escrevem a mensagem entre aspas representando `\n` a mudança de linha.■
- A chamada à função `scanf("%d",&n2)`, permite a introdução (leitura) de um valor, como inteiro decimal, que é guardado na variável `n2`.



Ambas as funções pertencem à biblioteca *standard* do C.

# Verificação: executar um programa em C – ambiente Unix

Visão optimista. . . ■

- Escrever o programa usando um editor de texto (por exemplo, Emacs) e guardar num ficheiro. Seja o ficheiro `comp.c`. O nome é arbitrário mas têm que ter a terminação `“.c”` ■

- Compilar o programa com o compilador `gcc` ou `cc` ■

```
$ gcc comp.c -o comp ■
```

O programa executável chama-se `comp` ■

(Sem a opção `-o` o executável é `a.out`) ■

- Executar o programa (comando): numa shell basta escrever o nome dele

```
$ comp
Introduz o primeiro inteiro
23
Introduz o segundo inteiro
45
45 é maior do que 23
$
```

Para verificar a correção será necessário provar matematicamente que em todas as condições a solução é a correcta. Na prática, pelo menos verificar para cada tipo de dados...


[ePD94, Cap. 2.1-4]

**Problema 2.** *A qualidade do ar é medida por um índice numérico. Se o índice for menor que 35 a qualidade do ar é Boa. Se esse valor for entre 35 e 60 é Má. Se esse valor for maior do que 60, a qualidade é Péssima.*



Não esquecer as etapas...

1. Ok...

2. Começemos por analisar um só valor:

(a) Ler um índice

(b) Escrever a mensagem apropriada, baseada no valor do índice. 

3. **Algoritmo**

Ler índice

Se índice < 35 então escrever "Boa"

senão se índice >= 35 e índice <= 60 escrever "Má"

senão escrever "Péssima"



#### 4. Codificação em C

```
#include <stdio.h>

main() {
    int indice;
    printf("Indice da qualidade do ar:");
    scanf("%d",&indice);
    if (indice < 35)    printf("Boa\n");
    else if( indice >= 35 && indice <= 60)    printf("Má\n");
    else printf("Péssima\n");
}
```

#### 5. Verificar que as condições são avaliadas correctamente.

**Problema 3.** *Analisar a qualidade do ar ao longo de 30 dias. Deve-se determinar o número de dias com cada classe de qualidade do ar e qual a média aritmética ao fim do mês.*

■

1. ...■

2. Considerar alguns valores: 30, 45, 40, 38, 49, 55, 40, 34,... e resolver o problema à mão. ■ Descrever o que foi feito: ■

*Para cada dia, ler um índice, determinar a sua classe e incrementar o número de dias com essa classe. Ir somando os valores lidos para determinar a média no fim.*

■

Ver que variáveis são necessárias:■

índice para o índice■

dias para contar os dias 1 a 30...■

boa para contar quantos os dias com qualidade do ar "Boa"; começa em 0.■

ma para contar quantos os dias com qualidade do ar "Má", começa em 0.■  
pessima para contar quantos os dias com qualidade do ar "Péssima", começa em 0.■  
soma para o cálculo da média é necessário somar todos os valores dos índices; começa em 0.■  
media para a média ■

### 3. Algoritmo

```
dias = 1, boa = 0, ma = 0, pessima = 0, soma = 0
Enquanto dias <= 30 faça
  Ler indice
  Se indice < 35 então boa = boa + 1
  senão se indice >=35 e indice <= 60 ma = ma + 1
    senão pessima = pessima + 1
  soma = soma + indice
  dias = dias + 1
media = soma / 30
Escrever boa, ma, pessima, media
```

## 4. Codificação em C

```
#include <stdio.h>
#define DIAS 30
main() {
    int indice, dias = 1, boa = 0, ma = 0, pessima = 0, soma = 0;
    double media;

    printf("Indices da qualidade do ar:\n");
    while (dias <= DIAS) {
        scanf("%d",&indice);
        if (indice < 35)    boa= boa + 1;
        else if( indice >= 35 && indice <= 60)    ma = ma + 1;
        else pessima = pessima + 1;
        soma = soma + indice;
        dias = dias + 1;
    }
    media = (float) soma / DIAS;
```

```

printf("\nQualidade || Número de dias\n");
printf("-----\n\n");
printf("    Boa          %4d\n",boa);
printf("    Ma           %4d\n",ma);
printf(" Pessima        %4d\n\n",pessima);
printf("Em média a qualidade foi ");
if (media < 35 ) printf("Boa\n");
else if( media >= 35 && media <= 60)  printf("Má\n");
    else printf("Péssima\n");
}

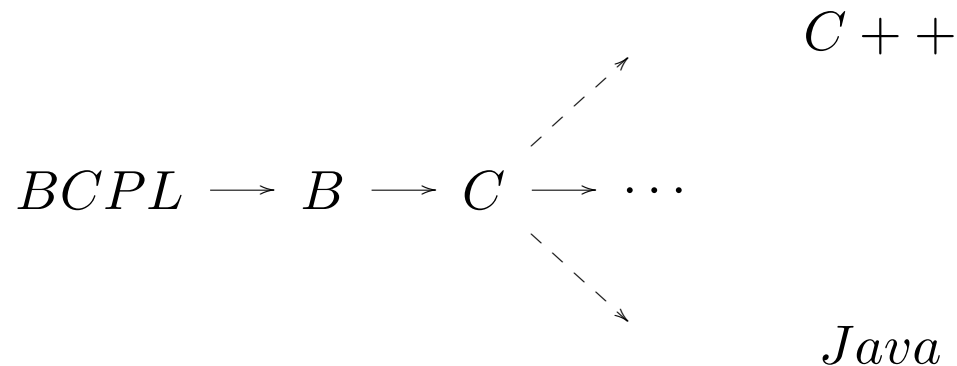
```

## 5. Verificação

**Exercício 2.1.** *Modificar o programa anterior para que o número dias introduzido possa ser um inteiro qualquer dado. ◇*

[ePD94, Cap. 3.1-10]

# A História da Linguagem C



C é linguagem de alto nível mais perto da linguagem máquina em termos de eficiência... actualmente existem compiladores e optimizadores de código que evitam o uso de linguagens assembly.

É usada essencialmente para tarefas em que a velocidade é crítica: interfaces a sistemas de operação; todo o tipo de algoritmos com “complexidade” alta.

Linguagem	Autores	Data	Descrição
BCPL	Martin Richards	1967	Para escrita de sistemas de operação e compiladores. Não tipificada.
B	Ken Thompson	197?	Usada para as primeiras versões do UNIX, num DEC-PDP-7. Não tipificada
C	D. Ritchie	1972	Usada para o UNIX, num DEC-PDP-11. Pretende ser portátil. Tipificada.
C	+ B. Kernighan	1978	C tradicional. Livro K&R (1ed).
ANSI C	X3J11	1989	C standard. Definição não ambígua e independente da máquina. Livro K&R (2ed).

# Tipos de dados simples do C

■  
`int` números **inteiros** dentro de um certo intervalo (p.e de 32 bits). ■ O tipo `int` pode ainda subdividir-se usando os modificadores: `short`, `long` e `unsigned`. Os dois primeiros relacionam-se com o tamanho de inteiros e o último permite considerar só números inteiros positivos.■

`float` números **racionais** em vírgula flutuante (precisão simples)■

`double` números racionais em vírgula flutuante (precisão dupla)■

`char` valores que representam **caracteres**, segundo um dado código (p.e código ASCII)■

`enum` valores num **conjunto** ordenado definido pelo utilizador (enumerações)■

Podem ser constantes ou variáveis.■



# Constantes

Exemplo	Tipo
56215	inteira (em base 10)
233	inteira (em base 10)
0x1f	inteira (em base 16)
037	inteira (em base 8)
12.67	racional
3.489e-7	racional (notação científica)
'f'	caracter
'\n'	caracter especial mudança de linha
'\t'	caracter especial tabulação
'\021'	caracter expresso no código ASCII
"shgf-&%8"	sequência de caracteres (string)
{NAO,SIM}	enumeração associa 0 a NAO e 1 a SIM

# Constantes Simbólicas

`#define nome expressão`

O pré-compilador substituí todas as ocorrências dessas constantes pelo respectivo valor.

**Problema 4.** *Tabelar a função  $\sin(x)$  entre 0 e 90 graus*

```
#include<stdio.h>
#define MIN      0
#define MAX      90
#define INCRE     5
#define PI       3.1415926
#define FCONV    PI/180
main() { /* tabela da função "seno" entre 0..90 graus */
    int x;
    for (x = MIN; x <= MAX; x += INCRE)
        printf("%3d %6.5f\n", x, sin(FCONV*x));
}
```

# Variáveis

Permitem guardar valores e são designadas por nomes. Os nomes começam por uma letra, podendo conter letras, dígitos e “\_”. Exemplos de nomes: `n`, `soma_10`, `TOTAL`.

Antes de ser usada uma variável tem de ser *declarada*. Uma declaração permite:

- Definir o nome da variável
- Definir o seu tipo
- Pode inicializar a variável

Exemplos:

```
int soma;  
int i = 0, n = MAX + 1;  
float media;  
char c = 'x';
```

# Expressões



Combinações com sentido de constantes, variáveis, chamadas a funções, operadores, parêntesis. ■

Exemplos: ■

55 + x

5

5 \*(sin(x) + 1)

■ x = 7 \* y + z/w + (z-1)

■ x + 1 != 4 || y > x + 2

■ x == 4 && y > 0

■ x >= 4 && !y > 0

■ x > 'a'

# Operadores aritméticos

[ePD94, Cap. 2.5]

Os *operadores binários aritméticos* são +, -, \*, / e o operador módulo % (=resto da divisão inteira). O único operador aritmético unário é o -.

```
for(i = 5; i < 100; i = i + 5)
    if (i % 3 != 0) printf( "%d \n",i);
```

Se os operandos forem inteiros o operador / produz a divisão inteira.

```
int n = 4, q, r;
q = 30 / n; r = 30 % n;
```

Os operadores + e - têm a mesma precedência. Esta é menor que a precedência de \*, / e %, que por sua vez é menor que a do operador unário -. A associatividade, para estes operadores, é da esquerda para a direita.

# Operadores relacionais

[ePD94, Cap. 2.6]

==	<i>igualdade</i>
!=	<i>desigualdade</i>
>	<i>maior que</i>
>=	<i>maior ou igual</i>
<	<i>menor que</i>
<=	<i>menor ou igual</i>

O valor duma expressão que envolve estes operadores ou é 0 (*falso*) ou 1 (*verdade*).

Não confundir a igualdade == com a atribuição = .

```
n = 1;  
if (n == 0) x = x + n; else x = 8;  
if (n != p) x = n; else x = n;
```

# Operadores lógicos

[ePD94, Cap. 4.10-4.11]

! negação  
&& conjunção (*e*)  
|| disjunção (*ou*)

- Uma negação tem o valor 1 se o argumento tiver o valor 0 e, 0 caso contrário.
- Uma conjunção tem o valor 1 se e só se todos os seus argumentos forem não nulos.
- Uma disjunção tem o valor 1 se e só se algum dos seus argumentos for não nulo.

Exemplos onde se supõe que  $x$  vale 5 e  $y$  vale 0. Determina o valor da expressão.

Expressão	Valor
<code>!(x == 8)</code>	1
<code>x == 8    y &lt; 1</code>	1
<code>5 &amp;&amp; 3</code>	1
<code>5 == 3    3 == 0</code>	0

A precedência de `&&` é maior que a de `||`, e ambos têm menor precedência que os operadores relacionais.

As expressões seguintes são equivalentes:

`i < 10 && i != 3 || i >= 20`

`((i < 10) && (i != 3) ) || (i >= 20)`

O operador unário de negação `!` converte um operando não nulo em 0 e um zero em 1.

- `if (!num) num=1` é equivalente a `if (num == 0) num=1;`
- `if (!(x==0)) x=0;` é equivalente a `if (x!=0) x=3;`



Nota que em C não existem expressões booleanas: uma expressão é verdadeira se o seu valor for diferente de zero, é falsa se o seu valor for zero.

# Leituras

[ePD94] (Cap. 2, 3.1-10, 4.10-11)

# Referências

[ePD94] H.M. Deitel e P.J. Deitel. *C:How to Program*. Prentice Hall International Editions, 2 edition, 1994.