Aula 8 – Comparações simples e comparações compostas

Comparações simples

São as comparações que usamos até agora dentro dos comandos **`if`** e **`while`**.
Comparam o valor de duas expressões aritméticas. A comparação é sempre verdadeira ou falsa.
O seu formato geral é:
`<expressão aritmética> <operador> <expressão aritmética>`
O `<operador>` pode ser: **`== , > , < , >= , <= , !=`**

Considere novamente o P9 já resolvido anteriormente.

P9) Dados 3 números imprimir o maior.

A seguinte construção facilitaria a solução:

```
Se (a>=b E a>=c) imprima a;
Se (b>=a E b>=c) imprima b;
Se (c>=a E c>=b) imprima c;
```

A novidade é o operador lógico E, ou seja, as duas comparações devem ser verdadeiras ao mesmo tempo.
Situações como esta são bastante comuns. A comparação deixa de ser simples para ser uma composição de duas comparações.
O operador lógico pode também ser **OU**, ou seja, uma das comparações pelo menos deve ser verdadeira.

Operadores Lógicos em C

Existem 3 operadores lógicos em C:
`&&` - E
`||` - OU
`!` - NÃO

Eles tem o significado usual como na lógica.
Os seus valores e prioridades encontram-se nas tabelas abaixo (V - verdadeiro; F - falso):

C1	C2	(C1 && C2)	(C1 || C2)	!C1
V	V	V	V	F
V	F	F	V	
F	V	F	V	V
F	F	F	F	

Operador	**Em C**	**prioridade**
E	&&	2
OU	||	3 (menos prioritário)
NÃO	!	1 (mais prioritário)

Vamos agora a solução do P9.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   /* variáveis usadas no programa */
   int a, b, c;

   /* leitura dos valores */
```

```
   printf("entre com os 3 valores:");
   scanf("%d%d%d", &a, &b, &c);

   /* verifica se a é o maior */
   if (a>=b && a>=c) printf("\nMaior:%d", a);
   /* verifica se b é o maior */
   if (b>=a && b>=c) printf("\nMaior:%d", b);
   /* verifica se c é o maior */
   if (c>=a && c>=b) printf("\nMaior:%d", c);
   system("PAUSE");
   return 0;
}
```

Exercícios com comparações compostas

1) Supondo `a=1, b=2 e c=3,` diga se são verdadeiras ou falsas as expressões abaixo:

```
(a<b) && (a<c)
(c<b) || (b>a)
(a==1) || (a<b) && (a<c)
(a >= 1) || (a == b) && b( > c)
(a+b > c) && (a == b - 1 || a!= b) && a + 2 == c
(a == 1) && b == 2 || c != 3
(a == b-1) || (a>b) && (c == b+1)
```

2) Refaça agora os problemas abaixo usando comparações compostas:

P10) Dados 3 números imprimi-los em ordem crescente (o primeiro menor ou igual ao segundo que é menor ou igual ao terceiro).

P10a) Idem imprimindo em ordem decrescente.

P11) Dados 3 números positivos verificar se são lados de um triângulo retângulo.

P12) Idem, verificando se são lados de algum triângulo, isto é, se o maior é menor que a soma dos outros dois.

Exemplo – consistência dos dados de entrada

Considere o seguinte problema:

P27) Dados N e M > 0 calcular o Máximo Divisor Comum entre N e M.

É necessária a consistência dos dados de entrada.
Vamos refazer apenas a parte inicial da consistência para exemplificar o uso de condição composta:

```
/* leitura de N e M */
printf("entre com N e M:");
scanf("%d", &N, &M);
if (N<=0 || M<=0) {
    printf("N e M devem ser > 0");
    system("PAUSE"); return 0;
}
```

Ou ainda, esperar que usuário digite corretamente:

```
   /* leitura de N */
   printf("entre com N e M:");
   scanf("%d", &N, &M);

   /* esperar até que o usuário digite N e M corretos */
   while (N <= 0 || M<=0) {
        /* nova leitura de N */
        printf("\nN e M devem ser > 0 \nentre com N e M:");
        scanf("%d", &N, &M);
   }
```

MDC novamente

Uma outra forma de calcular o MDC entre N e M, é escolher o menor e decrementá-lo até encontrar um divisor de ambos. Muito menos eficiente que o algoritmo de Euclides.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int N, M, mdc;
   /* leitura de N e M */
   printf("entre com N e M:");
   scanf("%d%d", &N, &M);

   /* esperar até que o usuário digite N e M corretos */
   while (N <= 0 || M<=0) {
        /* nova leitura de N */
        printf("\nN e M devem ser > 0 \nentre com N e M:");
        scanf("%d%d", &N, &M);
   }
   /* o primeiro candidato é o menor entre N e M */
   if (N>M) mdc = M;
   else mdc = N;
   /* decrementar o menor até que seja divisor de ambos */
   while (N % mdc != 0 || M % mdc != 0) mdc = mdc - 1;
   /* imprimir o resultado */
   printf("\nMDC entre %d e %d:%d", N, M, mdc);
   system("PAUSE");return 0;
}
```

P25a) Dados N e M > 0, imprimir todos os divisores comuns de N e M.