

Estruturas Básicas

Aula – Tópico 2

Problema 3

Exibir o maior número inteiro que pode ser representado no computador.



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int x;
    short int y;
    char a;
    unsigned char b;

    x = pow(2,31)-1;      // maior int possível
    y = pow(2,15)-1;      // maior short int possível
    printf("x = %d  y = %d\n",x,y);
    x = x + 1;
    y = y + 1;
    printf("x = %d  y = %d\n",x,y);
    /* -----
       Atribuir os maiores valores possíveis
       para as variáveis a e b.
       ----- */
    a = pow(2,7)-1;
    b = pow(2,8)-1;
    printf("a = %d  b = %d\n",a,b);
    a = a + 1;
    b = b + 1;
    printf("a = %d  b = %d\n",a,b);
    system("PAUSE");
    return 0;
}
```

Qual o maior número inteiro?

- Para o compilador **GCC**, números inteiros são representados usando-se 32 bits (4 bytes).
- Como o bit mais significativo representa o sinal, sobram 31 bits para representar o valor do número (complemento-de-2). O **maior inteiro** será:

01111111111111111111111111111111 = $2^{31} - 1 = 2147483647$

- Como assim?
 - Com **n** bits, podemos representar **2^n** números distintos, sendo o maior número **$2^n - 1$** . Exemplo: para $n = 2$, temos 4 números possíveis, sendo 3 o maior número.

Complemento-de-2

- **Atenção!**

- Na representação em complemento-de-2 existe sempre um valor negativo a mais.

0000	0001	0010	0011	0100	0101	0110	0111
0	+1	+2	+3	+4	+5	+6	+7
1000	1001	1010	1011	1100	1101	1110	1111
-8	-7	-6	-5	-4	-3	-2	-1

Menor inteiro

- Assim, o menor valor inteiro representável não será: -2147833647, mas sim -2147833648.
- Como assim?
 - Com n bits, o menor número representável será -2^{n-1} . Exemplo: para $n = 4$, o menor número representável é $-2^3 = -8$.
- Portanto, as variáveis do tipo `int` poderão armazenar valores no intervalo de -2147833648 a 2147833647.

Modificadores de tipo

- A linguagem C define alguns modificadores de tipo. Alguns deles são: `short`, `long`, `unsigned`.
- Um modificador de tipo altera o intervalo de valores que uma variável pode armazenar.
- Ao tipo `float` não se aplica nenhum dos modificadores, ao tipo `double` aplica-se apenas o modificador `long` e ao tipo `char` aplica-se somente o tipo `unsigned`.
- O modificador de tipo `short` instrui o compilador a representar valores inteiros usando 16 bits.
- Logo, uma variável `short int` pode armazenar valores inteiros no intervalo: -2^{15} a $2^{15} - 1$.

Modificadores de tipo

- Para as variáveis do tipo `char`, o compilador reserva 8 bits.
- Assim, variáveis do tipo `char` podem armazenar valores inteiros no intervalo -2^7 a $2^7 - 1$.
- O modificador de tipo `unsigned` instrui o compilador a não considerar o primeiro bit como sinal. Assim, variáveis `unsigned char` podem representar valores positivos maiores. O maior valor será: $2^8 - 1$.

Modificadores de tipo

No programa `p03.c` são atribuídos os maiores valores possíveis às variáveis `x` e `y`.

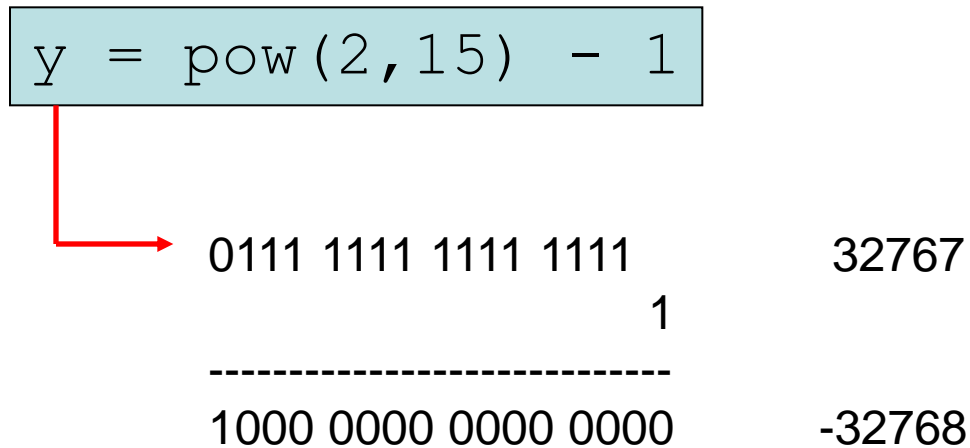
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int x;
    short int y;
    char a;
    unsigned char b;

    x = pow(2,31)-1; // maior int possível
    y = pow(2,15)-1; // maior short int possível
    printf("x = %d y = %d\n",x,y);
    x = x + 1;
    y = y + 1;
    printf("x = %d y = %d\n",x,y);
    /* -----
       Atribuir os maiores valores possíveis
       para as variáveis a e b.
       ----- */
    a = pow(2,7)-1;
    b = pow(2,8)-1;
    printf("a = %d b = %d\n",a,b);
    a = a + 1;
    b = b + 1;
    printf("a = %d b = %d\n",a,b);
    system("PAUSE");
    return 0;
}
```


Modificadores de tipo

- Em seguida, os valores das variáveis são **incrementados de 1**.
- O que acontece então?
- Ocorre um extravasamento (**overflow**)!
Exemplo: considere a variável **y**.



Avaliação de expressões aritméticas

- Os operadores aritméticos disponíveis na linguagem C são:

Operador	Operação
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão

Conversão implícita de tipo

- Na avaliação de expressões aritméticas, estas operações são realizadas sempre entre operandos de mesmo tipo.
- Ou seja, o resultado da operação terá o mesmo tipo que os operandos.
- Caso haja valores inteiros e em ponto flutuante em uma expressão, haverá uma conversão implícita de tipo de `int` para `float`, sempre que necessário.

Prioridade de execução das operações

- Porque as operações aritméticas devem ser feitas entre operandos do mesmo tipo?
 - As representações dos números inteiros e dos números de ponto flutuante são diferentes.
- Ou seja, embora 1 e 1.0 são valores iguais, eles tem representações diferentes no computador.
- **Prioridade de execução das operações:**
 - 1) expressões entre parênteses
 - 2) multiplicação, divisão e resto da divisão (da esquerda para a direita)
 - 3) operações de soma e subtração (da esquerda para a direita).

Prioridade de execução das operações

- Exemplo: $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja: $a = 1.5, b = 4, c = 2, d = 3, e = 1.2, f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 ^a	$(c + d)$	$(2 + 3) = 5$	Não
2 ^a	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não
3 ^a	$(a * 1^a)$	$(1.5 * 5) = 7.5$	Sim (5 para 5.0)
4 ^a	$(b * 2^a)$	$(4 * 5.5) = 22.0$	Sim (4 para 4.0)
5 ^a	$3^a / 4^a$	$7.5 / 22.0 = 0.341$	Não
6 ^a	$v1 = 5^a$	$v1 = 0.341$	Não

Conversão explícita de tipos

- É preciso **muito cuidado com a divisão inteira** (divisão entre operandos inteiros).
- O resultado da divisão inteira é sempre um número inteiro. Assim, se necessário, pode-se usar **uma conversão explícita de tipo** (*type casting*).

```
int a = 10, b = 3;
```

```
int c;
```

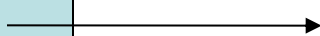
```
float d;
```

```
c = a / b;
```



c = 3

```
d = (float) a / b;
```



d = 3.333333

Conversão explícita de tipos

- **Atenção!**

- Observe que os resultados de:

```
d = (float) a / b;
```

(1)

e

```
d = (float) (a / b);
```

(2)

são totalmente diferentes!

- Em (1), primeiro realiza-se primeiro a conversão explícita de tipo (**a** torna-se 10.0) e, em seguida, realiza-se a divisão. Logo: **d = 3.333333**.
- Em (2), primeiro divide-se **a** por **b** e, em seguida, se faz a conversão explícita de tipo. Logo: **d = 3.0**.

Formatação de valores numéricos

- Além de especificar o número de casas decimais, um **tag** pode especificar o número total de caracteres (incluindo o sinal e o ponto decimal).
- Assim, o tag **%8.3f** significa: “exibir um valor de ponto flutuante com oito caracteres no total e com três casas decimais”.
- Se for necessário, será acrescentado o caractere ‘ ’ (espaço) à esquerda do valor para completar o tamanho total.

Formatação de valores numéricos

- Exemplo:

Valor	Tag	Valor exibido
pi = 3.14159	%5.3f	3.142
	%8.3f	3.142
raio = 2.0031	%5.3f	2.003
	%.6f	2.003100
area = 2*pi*raio	%5.3f	12.586
	%6.3f	12.586
	%7.3f	12.586
	%e	1.258584e+001
	%E	1.258584E+001
	%12.3e	1.259e+001

Formatação de valores numéricos

- A formatação de valores pode ser feita também para números inteiros.
- Exemplo:

Valor	Tag	Valor exibido
3	%d	3
	%5d	3
	%01d	3
	%05d	00003

Endereços de variáveis

- Uma **variável** representa um **nome simbólico** para uma posição de memória.
- Cada posição de memória de um computador possui um **endereço**. Logo, o endereço de uma variável é o endereço da posição de memória representada pela variável.

- Exemplo:

Operador para obtenção
do endereço da variável

Endereço no **sistema
hexadecimal**

Exibe:

```
int x = 3;  
printf("%d    %p", x, &x);
```

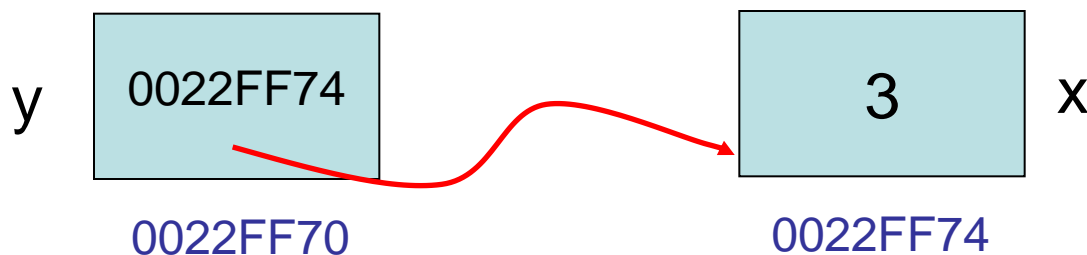
3 0022FF74

Endereços de variáveis

- Note que o endereço de uma variável é um valor. Logo, uma variável pode armazenar um endereço.
- Uma variável que armazena um endereço de memória é conhecida como **ponteiro** (*pointer*).
- Daí o porque do tag usado para exibir endereços de memória ser **%p**.

Endereços de variáveis

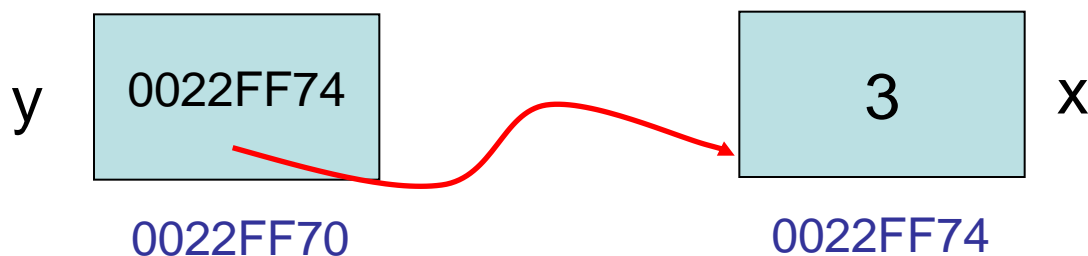
- Exemplo: suponha que y armazene o endereço **0022FF74** de uma posição de memória representada pela variável x . E ainda, que x contenha o valor inteiro 3.
- Esquemáticamente, podemos representar:



- Diz-se que y é um **ponteiro** para x , ou que y **aponta** para x .

Endereços de variáveis

- Qual é o tipo da variável *y*?
 - Para **declarar um ponteiro** é preciso saber para qual tipo de valor este ponteiro irá apontar.
 - Exemplo do caso anterior:



- Neste caso, o ponteiro aponta para um valor inteiro. Assim, diz-se que o tipo de *y* é **int ***.
- A declaração da variável *y* será:

```
int *y;
```

Indica que *y* é um ponteiro (para int, no caso)

Sistema hexadecimal

- O Sistema Hexadecimal (base 16) é o mais usado para representar endereços de memória.
- Grande poder de compactação: consegue representar 1 byte com apenas 2 dígitos!
- Ou seja, cada 4 bits são representados por um único algarismo hexadecimal.
- Neste sistema são utilizados 16 algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Sistema hexadecimal

- A tabela abaixo lista a correspondência entre os sistemas **binário**, **decimal** e **hexadecimal**.

Hexa	Decimal	Binário
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Qual o endereço armazenado em **y** no exemplo anterior, codificado em binário?

0022FF74



0000 0000 0010 0010 1111 1111 0111 0100

Conversão entre sistemas de numeração

- Para converter um valor no sistema hexadecimal para o correspondente valor no sistema binário e vice versa, o que devo fazer?
 - Consulte a tabela exibida na transparência anterior.
- Exemplos:
 - $(1267)_{16} = (0001\ 0010\ 0110\ 0111)_2$
 - $(1010\ 0010)_2 = (A2)_{16}$
 - $(1\ 0100)_2 = (14)_{16}$

Deve-se separar o número binário em blocos de 4 dígitos, da direita para a esquerda:

0001 0100

Conversão entre sistemas de numeração

- Para converter um valor no sistema hexadecimal para o correspondente valor no sistema decimal e vice versa, o que devo fazer?
- Exemplo:
 - $(ABAFA)_{16} = (703226)_{10}$

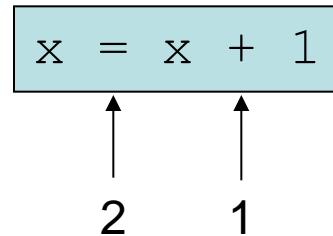
Casa	Valor da Casa	Lógica	Cálculo	Valor Decimal
5	A	$10 * (16 ^ 4)$	$10 * 65536 =$	655 360
4	B	$11 * (16 ^ 3)$	$11 * 4096 =$	45 056
3	A	$10 * (16 ^ 2)$	$10 * 256 =$	2 560
2	F	$15 * (16 ^ 1)$	$15 * 16 =$	240
1	A	$10 * (16 ^ 0)$	$10 * 1 =$	10
			Soma	703 226

- $(4711)_{10} = (1267)_{16}$

Número Decimal	Base	Resultado	Inteiro	Ajuste do Resto	Resto
$4711 / 16 =$		294,4375	294	$0,4375 * 16 =$	7
$294 / 16 =$		18,375	18	$0,375 * 16 =$	6
$18 / 16 =$		1,125	1	$0,125 * 16 =$	2
$1 / 16 =$		0,0625	0	$0,0625 * 16 =$	1

Operadores de incremento e decremento

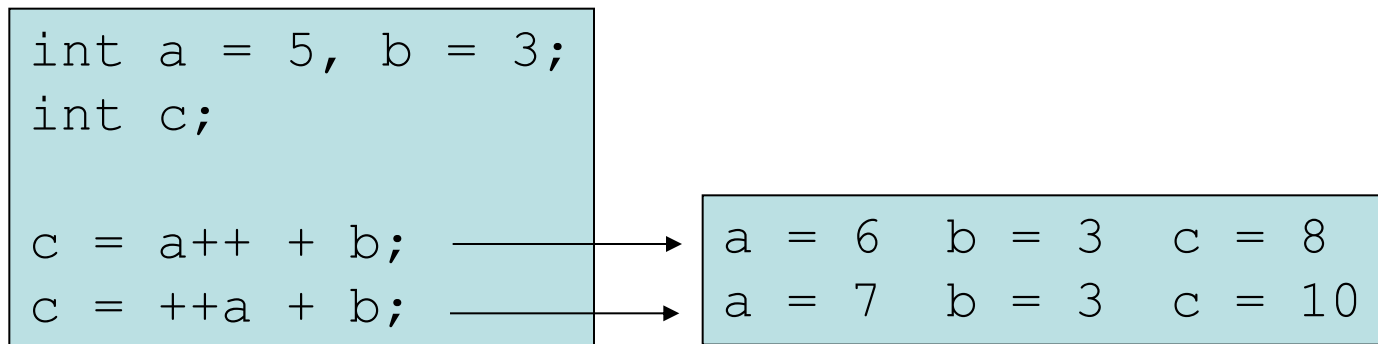
- Uma operação muito comum em programas de computador é **incrementar de 1** o valor da variável.
- Para fazer isso devemos:
 1. Somar 1 ao valor atual da variável;
 2. Armazenar o resultado na própria variável.



- Como a operação **incremento de 1** é muito comum, em C tem-se um operador especial: **`++`**.

Operadores de incremento e decremento

- Ao invés de escrevermos $x = x + 1$, podemos escrever: $x++$.
- Da mesma forma, para a operação decremento de 1: Em vez de $x = x - 1$, podemos escrever: $x--$.
- Os operadores $++$ e $--$ podem ser usados como **prefixo** ou como **sufixo** do nome da variável.



Operações combinadas com a atribuição

- As operações de **incremento** (++) e **decremento** (--) são exemplos de operações combinadas com a atribuição.
- Na linguagem C, sempre que for necessário escrever uma operação de atribuição da forma:

```
variavel = variavel operador expressao;
```

podemos combinar as operações.

Exemplos:

```
x = x + 5;
```

```
x = x - (a + b);
```

```
x = x * (a - b);
```

```
x = x / (x + 1);
```



```
x += 5;
```

```
x -= (a + b);
```

```
x *= (a - b);
```

```
x /= (x + 1);
```

Operações bit-a-bit

- Por uma questão de eficiência, a linguagem C dispõe de operações que podem ser feitas sobre a representação binária dos números inteiros.

Operador	Operação
< <	deslocamento para a esquerda
> >	deslocamento para a direita
&	conjunção bit-a-bit (<i>and</i>)
	disjunção bit-a-bit (<i>or</i>)
^	disjunção exclusiva bit-a-bit (<i>xor</i>)
~	negação bit-a-bit (<i>inverso</i>)

Operações bit-a-bit

- Tabela-verdade para cada operador.

and (&)

x \ y	0	1
0	0	0
1	0	1

or (|)

x \ y	0	1
0	0	1
1	1	1

xor (^)

x \ y	0	1
0	0	1
1	1	0

Operações bit-a-bit

Hexadecimal	Binário
0FF0	0000 1111 1111 0000
FF00	1111 1111 0000 0000
0FF0 << 4	1111 1111 0000 0000 = FF00
0FF0 >> 4	0000 0000 1111 1111 = 00FF
0FF0 & FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 0000 1111 0000 0000 = 0F00
0FF0 FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 1111 1111 1111 0000 = FFF0
0FF0 ^ FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 1111 0000 1111 0000 = F0F0
~ 0FF0	0000 1111 1111 0000 ----- 1111 0000 0000 1111 = F00F

Operações bit-a-bit

- Exemplos:

```
int a = 0x0FF0;  
int b = 0xFF00;  
int c;  
  
c = a << 4; printf("%04X << 4 = %04X\n", a, c);  
c = a >> 4; printf("%04X >> 4 = %04X\n", a, c);  
c = a & b;  printf("%04X & %04X = %04X\n", a, b, c);
```

Serão exibidos:

```
0FF0 << 4 = FF00  
0FF0 >> 4 = 00FF  
0FF0 & FF00 = 0F00
```

Problema 5

- Determine as raízes da equação $ax^2 + bx + c = 0$.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;

    delta = b*b - 4*a*c;
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :
                                                "nao possui raizes reais");

    if (delta >= 0)
    {
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
    }

    system("PAUSE");
    return 0;
}
```

Processamento condicional

- Todo programa na linguagem C inicia sua execução na primeira instrução da função **main**.
- As instruções são executadas **sequencialmente**, na ordem em que aparecem no texto.
- Muitas vezes, é necessário executar um conjunto de instruções **se uma condição for verdadeira** e, **caso contrário**, um outro conjunto de instruções.
- Quando um programa executa ou deixa de executar instruções com base no valor de uma condição, o programa realiza um **processamento condicional**.

Processamento condicional

- O programa `p05.c` realiza um processamento condicional.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
int main(int argc, char *argv[])
```

```
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;
```

```
    delta = b*b - 4*a*c;
```

```
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :  
        "nao possui raizes reais");
```

```
    if (delta >= 0)
    {
```

```
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
```

```
    }
```

```
    system("PAUSE");
```

```
    return 0;
```

```
}
```

Estas instruções serão executadas
somente se $\text{delta} \geq 0$.



Processamento condicional

- Para executar um processamento condicional, um programa precisa utilizar o comando **if**.
- Todo comando **if** requer uma **condição**. O valor de uma condição pode ser **verdadeiro** ou **falso**.
- Em C, não existe um tipo de dados específico para representar valores lógicos (**V** ou **F**).
- Qualquer valor diferente de zero é interpretado como verdadeiro, enquanto zero é falso.

Operadores relacionais

- Para escrever condições, são utilizados os **operadores relacionais** e os **operadores lógicos**.

Operador	Significado
>	Maior do que.
<	Menor do que.
>=	Maior do que ou igual a.
<=	Menor do que ou igual a.
==	Igual a.
!=	Diferente de.

Condição	Valor lógico
$(a \neq x)$	Verdadeiro.
$(a/2.0 == x)$	Verdadeiro.
$(a/2 == x)$	Falso.
$(a/x < 2)$	Falso.
(a)	Verdadeiro.
$(a - 2*x)$	Falso.

```
int a = 3; float x = 1.5;
```

Operadores lógicos

- Os operadores lógicos permitem combinar várias condições em uma única expressão lógica.

Operador	Significado
&&	Conjunção lógica (“and”)
	Disjunção lógica (“or”)
!	Negação lógica (“not”)

Expressão	Valor Lógico
$((a/2 == x) \ \&\& \ (a > 2))$	Falso.
$((x \leq a) \ \&\& \ (a \geq 2*x))$	Verdadeiro.
$(!(a/3 \leq x))$	Falso.
$(a \ \&\& \ x)$	Verdadeiro.
$((a - 2*x) \ \ (x < a/2))$	Falso.

```
int a = 3; float x = 1.5;
```

Operador condicional

- O operador condicional na linguagem C tem a seguinte sintaxe:

```
(condição) ? resultado-se-condição-verdadeira : resultado-se-condição-falsa
```

- Os resultados podem ser de qualquer tipo (int, float, char, double) e mesmo strings.
- Exemplos:

```
(b != 0) ? a/b : 0  
(peso <= 75) ? "ok" : "deve emagrecer"
```


Operador condicional

- O operador condicional pode ser usado em atribuições.
- Exemplo:

```
float nota1 = 5.0, nota2 = 4.0;  
  
media = ((nota1 >= 3) && (nota2 >= 5)) ?  
        (nota1 + 2*nota2)/3 :  
        (nota1 + nota2)/2;
```

↓
media recebe o valor 4.5

Qual seria o valor de média se:
float nota1 = 5.0;
float nota2 = 6.5;

Operador condicional

- No programa `p05.c`, o operador condicional é usado dentro da função `printf`.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int a = 2, b = 3, c = 1;
    float delta, x1, x2;

    delta = b*b - 4*a*c;
    printf("A equacao %s\n", (delta >= 0) ? "possui raizes reais" :
                                                "nao possui raizes reais");
    if (delta >= 0)
    {
        printf("As raizes sao %s\n", (delta > 0) ? "diferentes" : "iguais");
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf("Raiz x1 = %f\n", x1);
        printf("Raiz x2 = %f\n", x2);
    }

    system("PAUSE");
    return 0;
}
```

Atribuição e teste de igualdade

- **Atenção!**

- Um erro comum em linguagem C é usar o operador de atribuição (=) em vez do operador relacional (==) em condições que testam igualdade.

```
int fator = 3;
if (fator == 1)
{
    printf("O fator e' unitario\n");
}
printf("fator = %d\n", fator)
```

Imprime:

fator = 3

pois:

(fator == 1) é falso!

```
int fator = 3;
if (fator = 1)
{
    printf("O fator e' unitario\n");
}
printf("fator = %d\n", fator)
```

Imprime:

O fator e' unitario

fator = 1

pois:

(fator = 1) é verdadeiro!