



# **Introdução à Programação**

## **Capítulo III**

### **Tipos, Operadores e Expressões**

*Licenciatura em Engenharia Informática*

# Variáveis

Variáveis: servem para guardar informação que pode variar ao longo da execução de um programa.

- Está associada a uma posição de memória;
- Deve ser definida antes da sua utilização. (e antes de qualquer instrução)

Definição de variáveis:

*<tipo> <var1> [, <var2>, <var3>, ...];*

Inicialização de variáveis:

- Deve ser inicializada antes da sua utilização  
(a menos que, pelas características, o seja implicitamente)

*<var1> = [ <expressão> | <valor> ];*

# Nomes de variáveis

Regra do bom senso: é aconselhável escolher nomes que estejam relacionados com o papel que a variável vai desempenhar.

## Regras para formar nomes:

- constituídos por letras, dígitos e *underscores* ;
- tem que começar por uma letra ou por um *underscore*;
- *case-sensitive* (letras maiúsculas diferentes das minúsculas);
- o tamanho do nome depende do compilador mas normalmente são permitidos até 32 caracteres (ou mais);
- existem *keywords* reservadas: têm um significado especial, logo não podem ser usadas como identificadores. (Exemplo: **if**, **int**, **char**, **else**, **for**)

## Exemplos de identificadores legais:

contador, quase, num\_pagina, float1, float2, \_float, \_under;  
conta, Conta, CONTA, ConTa      (todas estas são variáveis diferentes)

## Exemplo de variáveis ilegais:

4i, ve-tudo, if

## Existem palavras reservadas -> “KEYWORDS”

<b>asm</b>	<b>auto</b>	<b>break</b>	<b>case</b>	<b>cdecl</b>
<b>char</b>	<b>class</b>	<b>const</b>	<b>continue</b>	<b>default</b>
<b>delete</b>	<b>do</b>	<b>double</b>	<b>else</b>	<b>enum</b>
<b>_export</b>	<b>extern</b>	<b>far</b>	<b>float</b>	<b>for</b>
<b>friend</b>	<b>goto</b>	<b>huge</b>	<b>if</b>	<b>inline</b>
<b>int</b>	<b>interrupt</b>	<b>_loadds</b>	<b>long</b>	<b>near</b>
<b>new</b>	<b>operator</b>	<b>pascal</b>	<b>private</b>	<b>protected</b>
<b>publicc</b>	<b>register</b>	<b>return</b>	<b>_saverregs</b>	<b>_seg</b>
<b>short</b>	<b>signed</b>	<b>sizeof</b>	<b>static</b>	<b>struct</b>
<b>switch</b>	<b>template</b>	<b>this</b>	<b>typedef</b>	<b>union</b>
<b>unsigned</b>	<b>virtual</b>	<b>void</b>	<b>volatile</b>	<b>while</b>

algumas implementações consideram mais palavras reservadas.

# Tipos de dados

Os tipos de dados básicos em C são:

- ***int***
- ***float***
- ***double***
- ***char***

( há quem também considere void e ponteiros (<tipo> \*)

## Inteiros - ***int***

A variáveis deste tipo são usadas para armazenar valores que pertencem ao conjunto dos números naturais, positivos e negativos. Ex: **2**, **-345**, **+115**, **0**.

O tamanho (em *bytes*) varia de arquitectura para arquitectura, sendo os valores mais habituais **2** ou **4 bytes**.

Nº de bytes	Menor valor	Maior valor
<b>2</b>	-32 768	32 767
<b>4</b>	-2 147 483 648	2 147 483 647

O formato para leitura / escrita de inteiros é **%d**.

Podem ser usados **4 prefixos** distintos:

- **short** - inteiro pequeno (2 bytes)
- **long** - inteiro grande (4 bytes)
- **signed** - inteiro com sinal (nºs positivos e negativos)
- **unsigned** - inteiro sem sinal (apenas nºs positivos)

short int	int	long int
2	2	4
2	4	4

Tipo de variável	Nº de bytes	Valor mínimo	Valor máximo
int	2	-32 768	32 767
short int	2	-32 768	32 767
long int	4	-2 147 483 648	2 147 483 647
unsigned int	2	0	65 535
unsigned short int	2	0	65 535
unsigned long int	4	0	4 294 967 295

O formato para leitura / escrita de inteiros **short** e **long** deve ser precedido dos prefixos **h** (short) e **l** (long).

## Reais – *float* e *double*

As variáveis deste tipo são usadas para armazenar valores numéricos com parte fraccionária. São designadas por variáveis reais ou de vírgula flutuante. Ex: **3.14**, **0.0000024514**, **1.0**.

A diferença entre uma variável do tipo ***float*** e uma variável do tipo ***double***, é o nº de *bytes* que é reservado para armazenar o valor.

A dimensão do ***float*** é normalmente de **4 bytes** (precisão simples), enquanto que a do ***double*** é de **8 bytes** (precisão dupla).

Tipo de variável	Nº de bytes	Menor valor positivo	Maior valor positivo
<b>float</b>	<b>4</b>	<b><math>1.17 \times 10^{-38}</math></b>	<b><math>3.40 \times 10^{38}</math></b>
<b>double</b>	<b>8</b>	<b><math>2.22 \times 10^{-308}</math></b>	<b><math>1.79 \times 10^{308}</math></b>

O formato para leitura / escrita de números float é **%f**.

O formato para leitura / escrita de números double é **%lf**.

Relativamente ao tipo ***double*** é permitido usar o prefixo ***long*** (long double)

→ possibilita ainda maior precisão (raramente é usado).

## Caracteres – *char*

O tipo ***char*** permite armazenar numa variável deste tipo, **um único caracter**.

Um ***char*** é sempre armazenado num *byte*.

Deste modo, o número de caracteres possíveis de representar é **256** (0...255).

00000000
----------

- todos os *bits* a 0 (valor 0)

11111111
----------

- todos os *bits* a 1 (valor 255)

O formato para leitura / escrita de caracteres é **%c**.

Para saber a dimensão de um tipo em C pode usar

sizeof<expressão>

ou

sizeof(<tipo>)



Alguns caracteres especiais:

<b>\7</b>	<b>Bell (sinal sonoro)</b>
<b>\a</b>	<b>Bell (sinal sonoro)</b>
<b>\b</b>	<b>Backspace</b>
<b>\n</b>	<b>New line (mudança de linha)</b>
<b>\r</b>	<b>Carriage return</b>
<b>\t</b>	<b>Tabulação horizontal</b>
<b>\v</b>	<b>Tabulação vertical</b>
<b>\\</b>	<b>Character \</b>
<b>\'</b>	<b>Character ‘</b>
<b>\”</b>	<b>Character “</b>
<b>\?</b>	<b>Character ?</b>

## Formatos de Leitura / Escrita (Resumo)

Tipo	Formato	Observações
<i>char</i>	<b>%c</b>	Um único caracter
<i>int</i>	<b>%d ou %i</b>	Um inteiro (base decimal)
<i>int</i>	<b>%o</b>	Um inteiro (base octal)
<i>int</i>	<b>%x ou %X</b>	Um inteiro (base hexadecimal)
<i>short int</i>	<b>%hd</b>	Um inteiro <i>short</i> (base decimal)
<i>long int</i>	<b>%ld</b>	Um inteiro <i>long</i> (base decimal)
<i>unsigned short int</i>	<b>%hu</b>	Um inteiro <i>short</i> positivo
<i>unsigned int</i>	<b>%u</b>	Um inteiro positivo
<i>unsigned long int</i>	<b>%lu</b>	Um inteiro <i>long</i> positivo
<i>float</i>	<b>%f ou %e ou %E ou %G</b>	
<i>double</i>	<b>%lf ou %le ou ...</b>	(double=long float)

```
#include <stdio.h>
main()
{ /* CUIDADO com os caracteres de formatação*/
    double db;
    float fl;

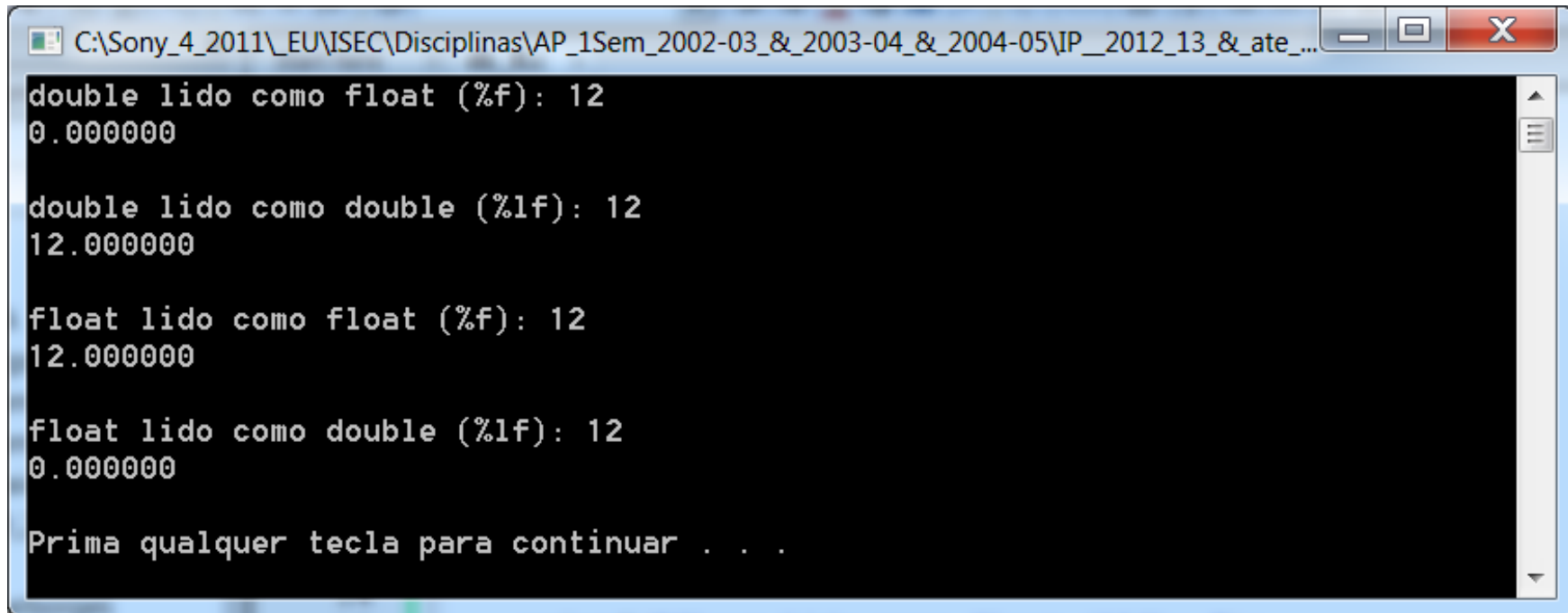
    printf("double lido como float (%%f): ");
    scanf("%f", &db);
    printf("%lf\n\n", db);

    printf("double lido como double (%%lf): ");
    scanf("%lf", &db);
    printf("%lf\n\n", db);

    printf("float lido como float (%%f): ");
    scanf("%f", &fl);
    printf("%f\n\n", fl);

    printf("float lido como double (%%lf): ");
    scanf("%lf", &fl);
    printf("%f\n\n", fl);
}
```

## Resultado de Execução:



```
C:\Sony_4_2011\EU\ISEC\Disciplinas\AP_1Sem_2002-03_&_2003-04_&_2004-05\IP_2012_13_&_ate_...  
double lido como float (%f): 12  
0.000000  
  
double lido como double (%lf): 12  
12.000000  
  
float lido como float (%f): 12  
12.000000  
  
float lido como double (%lf): 12  
0.000000  
  
Prima qualquer tecla para continuar . . .
```

## **Tipos de dados básicos em C - Tamanhos dos tipos**

- Em “limits.h” e “float.h” estão definidas constantes relativas aos tamanhos dos diversos tipos de dados;

<b>CHAR_BIT</b>	<b>CHAR_MAX</b>	<b>CHAR_MIN</b>
<b>INT_MAX</b>	<b>INT_MIN</b>	<b>LONG_MAX</b>
<b>LONG_MIN</b>	<b>SCHAR_MAX</b>	<b>SCHAR_MIN</b>
<b>SHRT_MAX</b>	<b>SHRT_MIN</b>	<b>UCHAR_MAX</b>
<b>UINT_MAX</b>	<b>ULONG_MAX</b>	<b>USHRT_MAX</b>
<b>...</b>		
<b>FLT_MAX</b>	<b>FLT_MIN</b>	<b>FLT_MIN_10_EXP</b>
<b>DBL_MAX</b>	<b>DBL_MIN</b>	<b>FLT_MAX_10_EXP</b>
<b>...</b>		

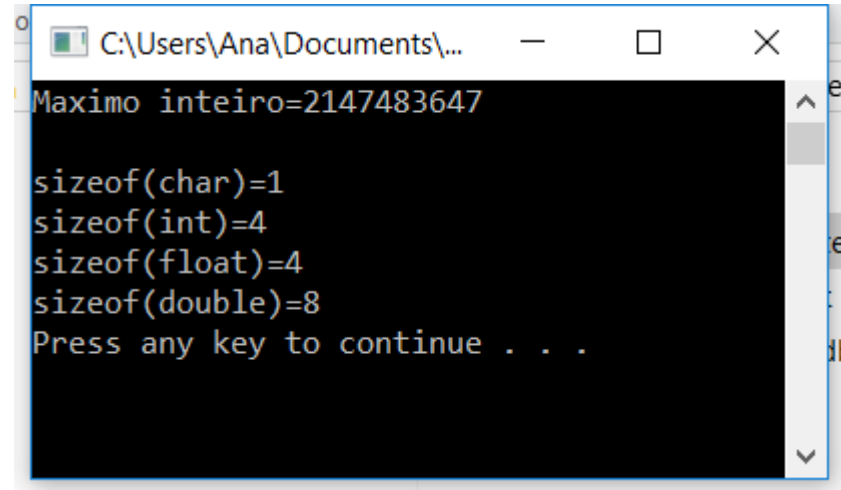
## Tipos de dados básicos em C - Tamanhos dos tipos

```
#include <stdio.h>
#include <limits.h>

main()
{
    int x=INT_MAX;

    printf("Maximo inteiro=%d\n\n", x);
    printf("sizeof(char)=%d\n", sizeof(char));
    printf("sizeof(int)=%d\n", sizeof(int));

    printf("sizeof(float)=%d\n", sizeof(float));
    printf("sizeof(double)=%d\n", sizeof(double));
}
```



```
Maximo inteiro=2147483647
sizeof(char)=1
sizeof(int)=4
sizeof(float)=4
sizeof(double)=8
Press any key to continue . . .
```

## Tipos de dados básicos em C - Tamanhos dos tipos

```
#include <stdio.h>
#include <float.h>
main()
{
    double xd=DBL_MAX;
    float xf=FLT_MIN;
    int ef=FLT_MIN_10_EXP;

    printf("Maximo double=%le\n\n", xd);
    printf("Maximo double=%lf\n\n", xd);

    printf("Minimo float=%e\n", xf);
    printf("Minimo expoente para base 10 no float=%d\n\n", ef);
}
```

## Resultado de Execução:

[illegible]



## Constantes

Uma constante não é mais do que um nome ao qual corresponde um valor fixo (que não se pode alterar ao longo de uma execução).

As constantes devem ser definidas fora das funções, de forma a serem “visíveis” por todo o código do programa. Normalmente a sua definição faz-se logo a seguir às linhas de *#include*.

Podem definir-se de duas formas:

- Através da palavra *const*.

***const* <tipo> <símbolo> = <valor>;**

- Através da directiva *#define*: ***#define* <símbolo> <valor>**

Por exemplo:

```
#include <stdio.h>

const int num = 10;

void main()
{
    int i;

    for(i=0; i<num; i++)
        printf("%d\n",i);
}
```

```
#include <stdio.h>

#define NUM 10

void main()
{
    int i;

    for(i=0; i<NUM; i++)
        printf("%d\n",i);
}
```

- Uma constante definida com **const** existe fisicamente numa determinada localização da memória.
- Uma constante definida com **#define** não existe fisicamente em memória. O seu valor é substituído ao longo do programa na fase de pré-processamento. As constantes assim definidas designam-se por **constantes simbólicas** e normalmente o seu nome escreve-se em maiúsculas.

# Tipos de dados básicos em C - Constantes

Têm implicitamente um tipo que é determinado pela maneira como são representadas

## ***Constantes inteiras:***

- *Podem ser representadas em 3 bases:*
  - constantes inteiras decimais (base 10)
  - constantes inteiras octais (base 8 - 077)
  - constantes inteiras hexadecimais (base 16 - 0xA1, ou 0X2C3)
- *Exemplos*

➤ 1066	int decimal
➤ 077	int octal
➤ 0xA1	int hexadecimal
➤ ...	
➤ 8623L (ou 8623l)	long int decimal
➤ 56h	short int decimal
➤ 12478ul (ou ...)	unsigned long int decimal
➤ ...	

### ***Constantes virgula flutuante:***

- *Exemplos*

- 2.0019 double
- -345e-2 double
- ...
- 86.23F (ou 86.23f) float
- 346.7e-03L (ou 346.7e-03l) long double
- ...

### ***Constantes caracter:***

- *Exemplos*

- 'P' (codificado como 80 em ASCII)
- 'T' (codificado como 84 em ASCII)
- ...
- '\n'
- ...

### ***Constantes string:***

- *Exemplos*

- "Uma constante string"
- "" (string vazia)
- ...

## Constantes em C - Constantes caracter:

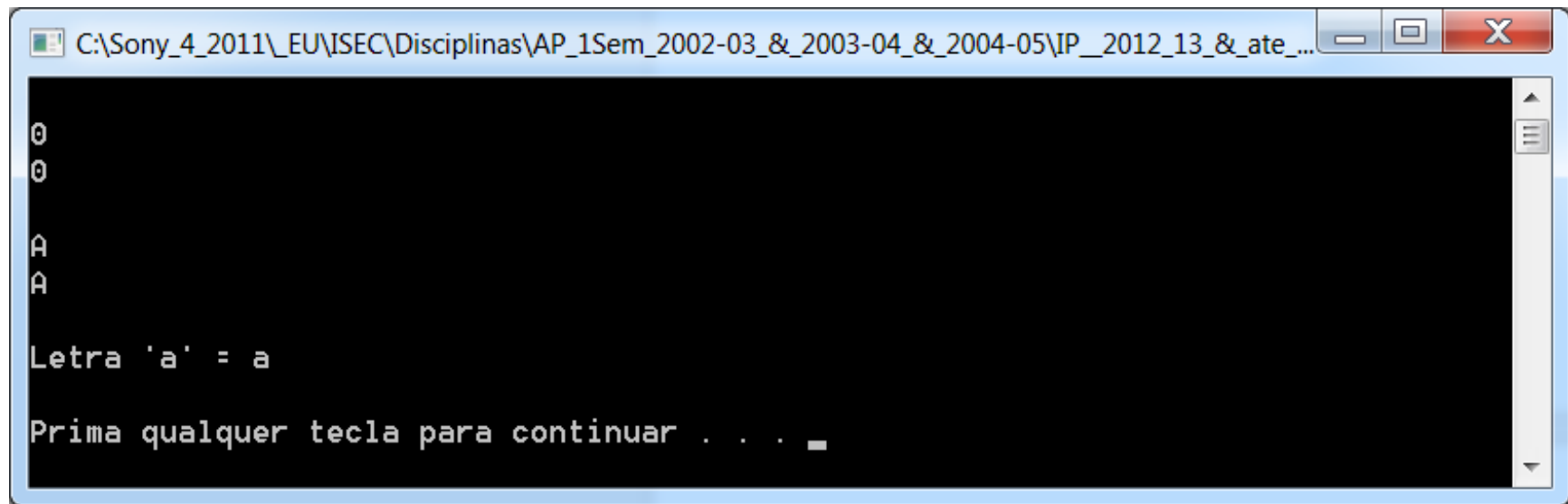
```
#include <stdio.h>

main()
{
/* Escreve o dígito '0' (código 48 em ASCII, decimal) 48=060(octal)=0x30(hexadecimal) */
    printf("\n\060\n");
    printf("\x30\n");
/* Escreve o caractere 'A' (código 65 em ASCII, decimal) */
    printf("\n\0101\n");
    printf("\x41\n");

/* Escreve o caractere 'a' (código 97 em ASCII, decimal) */
    printf("\nLetra 'a' = \x61\n\n");

    system("pause");
}
```

## Resultado de Execução:



```
C:\Sony_4_2011\EU\ISEC\Disciplinas\AP_1Sem_2002-03_&_2003-04_&_2004-05\IP_2012_13_&_ate...
0
0
A
A
Letra 'a' = a
Prima qualquer tecla para continuar . . . _
```

# Operadores

- *Já vimos anteriormente o operador atribuição (=) e alguns operadores aritméticos.*
- *O que é um:*
  - – Operador?
  - – Operando?
  - – Expressão?
- *Como se interpreta a tabela de precedência dos operadores?*
  - – Precedência dos operadores
  - – Associatividade dos operadores

## Tabela de Precedências

Operador	Associatividade
( ) [ ] -> .	→
! ~ ++ -- - (unário) + (unário) (casting) * (apontado) & sizeof	←
* / %	→
+ -	→
<< >>	→
< <= > >=	→
== !=	→
&	→
^	→
	→
&&	→
	→
?: (Operador Ternário)	←
= += -= *= /= %= <<=	←
>>= &= ^=  =	←
, (Vírgula)	→





## Associatividade dos Operadores em C:

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

```
main()
```

```
{      /* Associatividade dos operadores*/
```

```
    int a, b, c;
```

```
    a=1;
```

```
    b=2;
```

```
    c=3;
```

```
/* operador atribuição, associatividade da direita para a esquerda! */
```

```
    a=b=c=12;
```

```
    printf("\na=%d\tb=%d\tc=%d\n", a, b, c);
```

```
/* operador resto da divisão inteira (%), associatividade da esquerda para a direita! */
```

```
    a=4%3%2;
```

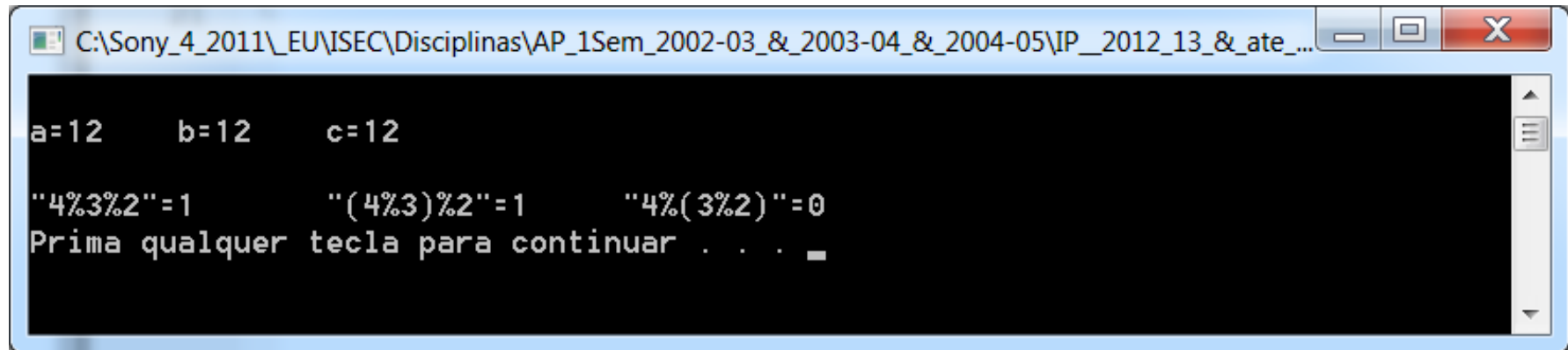
```
    b=(4%3)%2;
```

```
    c=4%(3%2);
```

```
    printf("\n\"4%%3%%2\"=%d\t\"(4%%3)%%2\"=%d\t\"4%%(3%%2)\"=%d\n", a, b, c);
```

```
}
```

## Resultado de Execução:



```
C:\Sony_4_2011\EU\ISEC\Disciplinas\AP_1Sem_2002-03_&_2003-04_&_2004-05\IP_2012_13_&_ate_...  
a=12    b=12    c=12  
"4%3%2"=1      "(4%3)%2"=1      "4%(3%2)"=0  
Prima qualquer tecla para continuar . . .
```

## Operadores aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto de divisão inteira
++	Incremento
--	Decremento

## Operadores lógicos e relacionais

Em C não existe nenhum tipo de dados específico para armazenar valores lógicos.

- O valor lógico **FALSO** é representado por **0** (zero).
- O valor lógico **VERDADE** não é 1, mas sim **qualquer valor diferente de 0** (zero). O valor 1 é apenas um dos valores possíveis.

## Operadores lógicos:

Operador	Significado
&&	E (AND)
	OU (OR)
!	NEGAÇÃO (NOT)

As expressões ligadas por && ou || são avaliadas da esquerda para a direita, terminando a avaliação logo que se possa concluir que o resultado global irá ser verdadeiro ou falso.

## Operadores relacionais:

Operador	Nome	Exemplo	Significado do Exemplo
==	Igualdade	a == b	a é igual a b ?
>	Maior que	a > b	a é maior que b ?
>=	Maior ou igual que	a >= b	a é maior ou igual que b ?
<	Menor que	a < b	a é menor que b ?
<=	Menor ou igual que	a <= b	a é menor ou igual que b ?
!=	Diferente de	a != b	a é diferente de b ?

# Operadores incremento e decremento

➤ O operador incremento `++` e o operador decremento `--` têm o efeito lateral de adicionar 1 ou subtrair 1 aos seus operandos:

- Podem utilizar-se de modo

- *pré-fixado* (`++i` ou `--j`)

- `n=++i` incrementa `i` antes de o atribuir a `n`

ou

- *pós-fixado* (`i++` ou `j--`)

- `n=i++` atribui `i` a `n` e só depois incrementa `i`

➤ Estes operadores só podem ser utilizados com variáveis  
`(i+j)++` é ilegal!

## Operadores de atribuição

- A atribuição pode combinar-se, por exemplo, com os operadores aritméticos binários:

➤ +, -, \*, /, %

– Se **op** for um destes operadores

- $\langle \text{exp1} \rangle \text{ op} = \langle \text{exp2} \rangle;$

é equivalente a

- $\langle \text{exp1} \rangle = \langle \text{exp1} \rangle \text{ op} \langle \text{exp2} \rangle;$

$i *= 2; \Leftrightarrow i = i * 2;$

$x /= y + 2; \Leftrightarrow x = x / (y + 2);$  (o que não é o mesmo que  $x = x / y + 2;$ )

- A atribuição (=), assim como os operadores de atribuição (+=, -=, ...) podem aparecer em expressões
  - O tipo de uma expressão de atribuição é o tipo do operando da esquerda e o valor é obtido após executada a atribuição.

## Expressão condicional

- Seja a seguinte expressão condicional:

$\langle \text{ex1} \rangle ? \langle \text{ex2} \rangle : \langle \text{ex3} \rangle$

- $\text{ex1}$  é avaliada
  - se for diferente de zero (verdade),  $\text{ex2}$  é avaliada e é o valor da expressão condicional;
  - se for igual a zero (falso),  $\text{ex3}$  é avaliada e é o valor da expressão condicional.

- Só uma de entre  $\text{ex2}$  ou  $\text{ex3}$  é avaliada.

- Por exemplo

$x = a < b ? a : b;$

atribui-se a  $x$  o menor dos valores entre  $a$  e  $b$

## Conversões de tipo

Quando um operador tem **operandos de tipos diferentes**, eles são **convertidos a um tipo comum**, antes da expressão final ser calculada. Em geral, as conversões automáticas transformam operandos de tipos mais “estreitos” em operandos de tipos mais “largos”.

As regras de conversão são as seguintes:

Se um dos operandos for **long double**, o outro é convertido para **long double** e o resultado da operação é **long double**.

Senão, se um dos operandos for **double**, o outro é convertido para **double** e o resultado da operação é **double**.

Senão, se um dos operandos for **float**, o outro é convertido para **float** e o resultado da operação é **float**.

Senão, se um dos operandos for **long int**, o outro é convertido para **long int** e o resultado da operação é **long int**.

Senão, se um dos operandos for **int**, quer o outro seja **char** ou **short int**, é convertido para **int** e o resultado da operação é **int**.



Se um dos operandos for **unsigned**, o outro é convertido para **unsigned** e o resultado é também **unsigned**.

Quando os membros de uma **expressão de atribuição** têm tipos diferentes, também ocorre uma **conversão de tipos**: o lado direito da expressão é convertido no tipo do lado esquerdo da mesma, que também é o tipo do resultado.

Para além das conversões automáticas referidas, as conversões podem também ser explicitamente requeridas pelo programador (**casting**):

**(tipo)** *expressão*

Exemplo:

```
int i=1,j=2;
```

```
float x;
```

```
x = (float)i / j;
```

## Tabela de Precedências

Operador	Associatividade
( ) [ ] -> .	→
! ~ ++ -- - (unário) + (unário) (casting) * (apontado) & sizeof	←
* / %	→
+ -	→
<< >>	→
< <= > >=	→
== !=	→
&	→
^	→
	→
&&	→
	→
?: (Operador Ternário)	←
= += -= *= /= %= <<=	←
>>= &= ^=  =	←
, (Vírgula)	→



# Exemplos

## Exemplo 1

---

```
/* Utilização de Operadores Aritméticos*/  
#include <stdio.h>  
#define METADE (1/2)  
  
void main()  
{  
    int a, b;  
    float x, y;  
  
    a = 2;  
    x = 0.5f;  
    y = a + x;  
    b = 7 / a;  
    printf("%d\n", a + b);  
    printf("%d\n", b % a);  
    printf("%f\n", y);  
    printf("%f\n", x/a);  
    printf("%f\n", METADE * y * b);  
}
```

### Resultado da execução:

```
5  
1  
2.500000  
0.250000  
0.000000
```

## Exemplo 1a

---

```
/* Utilização de Operadores Aritméticos (Versão Revista e Formatada) */

#include <stdio.h>

/* divisão de um real por um inteiro */
#define METADE (1.0f/2)

void main()
{
    int a, b;
    float x, y;

    a = 2;
    x = 0.5f;
    y = a + x;
    b = 7 / a;
    printf("%4d\n", a + b);
    printf("%4.4d\n", b % a);
    printf("%2.1f\n", y);
    printf("%4.4f\n", x/a);
    printf("%3.2f\n", METADE * y * b);
}
```

### Resultado da execução:

```
    5
0001
 2.5
0.2500
 3.75
```

## Exemplo 2

---

```
/* Operadores de Atribuição Compostos */

#include <stdio.h>

void main()
{
    double a = 1.0, b = 1.0, c = 1.0;
    int i = 4, j = 4;

    i *= 3 + 3;
    j = j * 3 + 3;
    printf("i = %d\nj = %d\n", i, j);

    a += b += c += 1.25;
    printf("a=%lf\nb = %lf\nc = %lf\n", a, b, c);
}
```

### Resultado da execução:

```
i = 24
j = 15
a = 4.250000
b = 3.250000
c = 2.250000
```

## Exemplo 3

---

```
/* Operadores de Incremento e Decremento */

#include <stdio.h>

void main()
{
    int a, b, c;

    a = 1;
    b = ++a;
    c = a++ + b--;

    printf("a = %d\nb = %d\nc = %d\n", a, b, c--);
}
```

**Resultado da execução:**

a = 3
b = 1
c = 4

## Exemplo 4

---

```
/* Operadores Relacionais e de Igualdade */

#include <stdio.h>

void main()
{
    int i = 2, j = 1, k;
    float x = 1.5f;

    k = i * j == 1;
    printf("%d\n", k);
    printf("%d\n", k > i < x);
    printf("%d\n", i == 2 == j);
    printf("%d\n", i - j != i > j % 4);
}
```

**Resultado da execução:**

0

1

1

0



## Exemplo 5

---

```
/* Operadores Lógicos */

#include <stdio.h>

void main()
{
    int a = 1, b = 2;
    int c = 3;

    printf("%d\n", c);
    printf("%d\n", !c);
    printf("%d\n", !!c);
    printf("%d\n\n", c);
    printf("Antes Av 1: a=%d b=%d\n", a, b);
    printf("Av 1: %d\n", a++ > 0 && b++ < 0);
    printf("Apos Av 1 : a=%d b=%d\n", a, b);
    printf("Av 2: %d\n", a++ <= 0 && b++ != 0);
    /* b nao foi incrementado */
    printf("Apos Av 2 : a=%d b=%d\n", a, b);
}
```

### Exemplo de execução:

```
3
0
1
3
Antes Av 1: a=1      b=2
Av 1: 0
Apos Av 1 : a=2      b=3
Av 2: 0
Apos Av 2 : a=3      b=3
```

## Exemplo 6

---

```
/* Avaliação de Expressões Complexas */
```

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

```
void main()
```

```
{
```

```
    int a, b, c;
```

```
    a = b = 0;
```

```
    printf("Inicio: a=%d b=%d c=%d\n", a==0, b!=0, c=0);
```

```
    ++a && ++b && c++;
```

```
    printf("Fim Av1: a=%d b=%d c=%d\n", a, b, c);
```

```
    --a || b-- || c--;
```

```
    printf("Fim Av2: a=%d b=%d c=%d\n", a, b, c);
```

```
    b = (a-5) >= 0 || !(c--) && (a=5);
```

```
    printf("Fim Av3: a=%d b=%d c=%d\n", a, b, c);
```

```
}
```

### Resultado da execução:

Inicio:	a = 1	b = 0	c = 0
Fim Av1:	a = 1	b = 1	c = 1
Fim Av2:	a = 0	b = 0	c = 1
Fim Av3:	a = 0	b = 0	c = 0

## Exemplo 7

---

```
/* Expressão Condicional */

#include <stdio.h>

void main()
{
    int num1, num2;

    printf("Diga 2 numeros: ");
    scanf("%d %d", &num1, &num2);
    printf("O maior deles é %d\n", num1>num2?num1:num2);
}
```

### Exemplo de execução:

```
Diga 2 numeros: 3 9
O maior deles é 9
```