



## Tipos & Variáveis

CAP ENGEL Diogo Silva  
[dasilva@academiafa.edu.pt](mailto:dasilva@academiafa.edu.pt)

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d\n.", n, cubo(n));  
  
    return 0;  
}
```

Uma função em C. Qual é o domínio e contradomínio desta função?

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d", n, cubo(n));  
  
    return 0;  
}
```

Uma função em C. Qual é o domínio e contradomínio desta função?



```
int cubo(int n) {
```

Tipo do *output*  
da função.

Tipo do *input*  
da função.

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()
```

# O que é um tipo?

```
printf("Introduza um valor: ");
```

```
scanf("%d", &n);
```

```
printf("O cubo de %d tem o valor %d", n, cubo(n));
```

```
return 0;
```

```
}
```

Uma função em C. Qual é o domínio e contradomínio desta função?



```
int cubo(int n){
```

Tipo do *output*  
da função.

Tipo do *input*  
da função.

Conjunto de valores;  
Literais;  
Operações;

O que é  
um **tipo**?

Conjunto de valores;  
Literais;  
Operações;

O C é uma linguagem **tipificada**.  
Todas as expressões, variáveis,  
parâmetros e resultados têm um  
tipo associado.

# O que é um **tipo**?

Conjunto de valores;  
Literais;  
Operações;

O C é uma linguagem **tipificada**.  
Todas as expressões, variáveis,  
parâmetros e resultados têm um  
tipo associado.

# O que é um tipo?

Quais os tipos do C?

**Tipos primitivos principais:**

- inteiros
- reais
- caracter

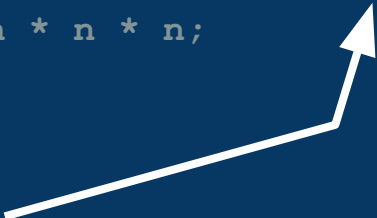
Programadores podem definir os seus  
próprios tipos.

Obrigatoriamente, existe uma função inteira chamada **main**, também conhecida por **função principal**.

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d\n.", n, cubo(n));  
  
    return 0;  
}
```





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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d\n.", n, cubo(n));  
  
    return 0;  
}
```

Obrigatoriamente, existe uma função inteira chamada **main**, também conhecida por **função principal**.

A função `main` é onde o programa começa a ser executado.

No C usamos **variáveis** para guardar dados na memória.

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d\n.", n, cubo(n));  
  
    return 0;  
}
```

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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()
```

```
{
```

```
    int n;
```

```
    printf("Introduza um valor inteiro\n");
```

```
    scanf("%d", &n); //pede valor ao u
```

```
    printf("O cubo de %d tem o valor d
```

```
    return 0;
```

```
}
```

No C usamos **variáveis** para guardar dados na memória. Podemos pensar na memória como apartados dos correios.



Cada caixa guarda 1 item.

Cada caixa pode ser identificada e  
acessada pelo seu endereço único.

Podemos colocar e retirar itens  
da caixa ou simplesmente ver o  
seu conteúdo.

No C usamos **variáveis** para  
guardar dados na memória.  
Podemos pensar na memória  
como apartados dos correios.



Cada caixa guarda 1 item.

Cada caixa pode ser identificada e acedida pelo seu endereço único.

Podemos colocar e retirar items da caixa ou simplesmente ver o seu conteúdo.

Todas as caixas são do mesmo tamanho. Diferem apenas pelo seu endereço. Se queremos maior capacidade precisamos de mais caixas.

Algumas caixas podem ser acedidas directamente, outras requerem autorização (memória gerida pelo SO).

No C usamos **variáveis** para guardar dados na memória. Podemos pensar na memória como apartados dos correios.



Cada caixa guarda 1 item.

Cada caixa pode ser identificada e acedida pelo seu endereço único.

Podemos colocar e retirar itens da caixa ou simplesmente ver o seu conteúdo.

Todas as caixas são do mesmo tamanho. Diferem apenas pelo seu endereço. Se queremos maior capacidade precisamos de mais caixas.

Algumas caixas podem ser acedidas directamente, outras requerem autorização (memória gerida pelo OS).

Quando temos muitos dados para guardar, temos de usar várias caixas adjacentes.



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

```
int cubo(int n){  
    return n * n * n;  
}
```

```
int main()  
{  
    int n;  
    printf("Introduza um valor inteiro:");  
    scanf("%d", &n); //pede valor ao utilizador  
    printf("O cubo de %d tem o valor de %d\n.", n, cubo(n));  
  
    return 0;  
}
```

No C usamos **variáveis** para guardar dados na memória.

A uma variável está associado um **tipo**, um **nome** e um espaço na memória (uma “caixa” com endereço único).

# VARIÁVEIS E ATRIBUIÇÕES

## Memória

'C'		FF	FF			FF	FF			00	02	16	53		
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115

→ zona de memória ocupada pela variável

declaração

inicialização / atribuição

char      letra

= 'c'

tipo

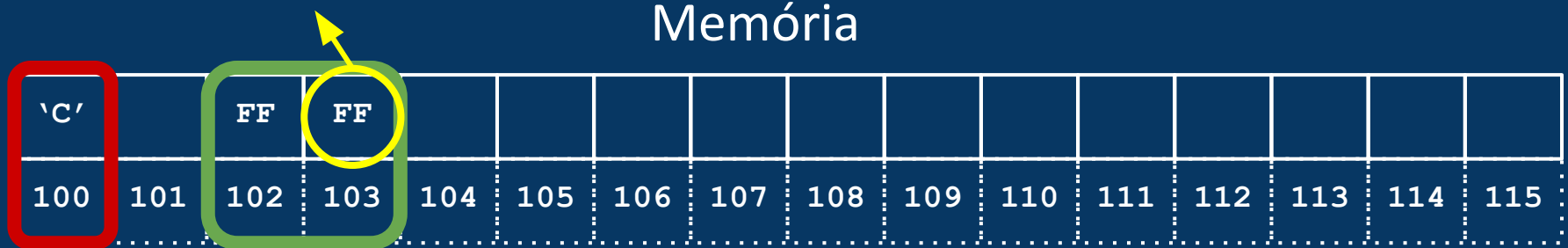
valor da variável



# VARIÁVEIS E ATRIBUIÇÕES

valor hexadecimal

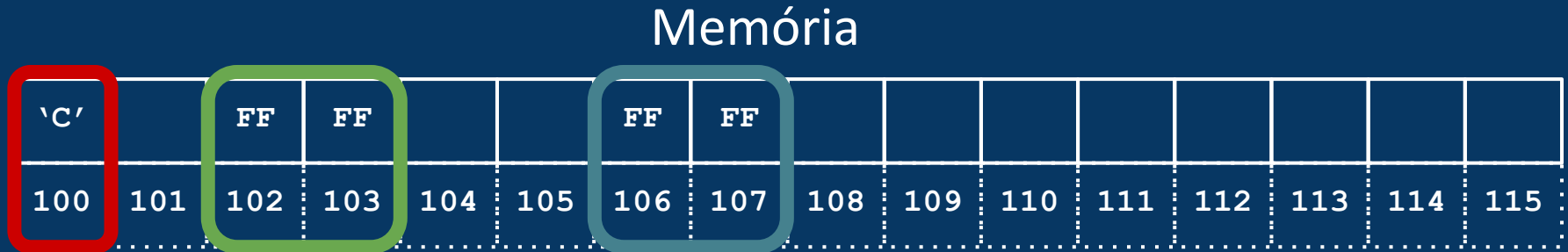
FF = 11111111 em binário



```
char    letra    = 'c'
```

```
short int num_negativo = -32768
```

# VARIÁVEIS E ATRIBUIÇÕES



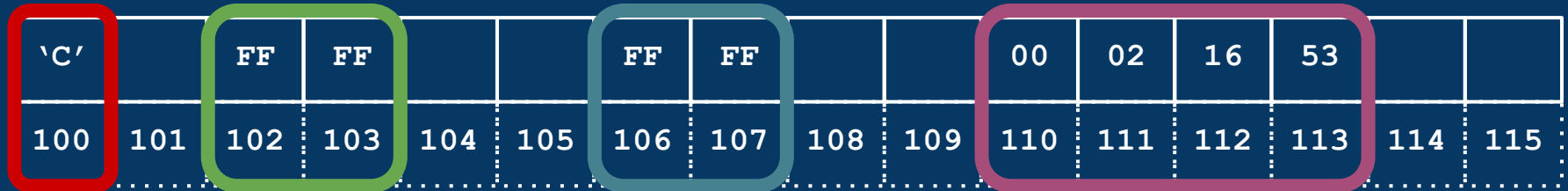
```
char    letra    = '\c'
```

```
short int num_negativo = -32768
```

```
unsigned short int num_positivo = 65535
```

# VARIÁVEIS E ATRIBUIÇÕES

## Memória



```
char    letra    = 'c'
```

```
short int num_negativo = -32768
```

```
unsigned short int num_positivo = 65535
```

```
int     nip      = 136787
```

```
sizeof(short int) -> 2
```

```
sizeof(int) -> 4
```

```
sizeof(float) -> 4
```

```
sizeof(double) -> 8
```

# IDENTIFICADORES

Nomear entidades:

- Constantes;
- Variáveis;
- Argumentos;
- Funções;
- Tipos;
- Etiquetas.

Em C, um identificador começa por uma letra ou sublinhado '\_' e continua com algarismos, letras e sublinhados, e.g.

```
valor  
_valor1  
x  
—
```

# Palavras reservadas

auto	break	case	char	const	continue	default	do	_Bool	restrict
double	else	enum	extern	float	for	goto	if	_Complex	
int	long	register	return	short	signes	sizeof	static	_Imaginary	
struct	switch	typedef	union	unsigned	void	volatile	while	Inline	

# COMENTÁRIOS

São ignorados pelo compilador

```
// comentario de 1 linha

/*
comentario longo, talvez a descrever
o que uma funcao faz - precisa de varias
linhas
*/
```

# TERMINADOR

Todas as instruções (mas não as expressões) são terminadas por “;”

```
double modulo(double n) {  
    if (n > 0) return n;  
    else return -n;  
}
```

# #DEFINE

Definição de constantes

```
#define PI  
3.14159265359
```

```
double p = PI / 2;
```

# #INCLUDE

Ficheiros fonte;

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

# #IF #ELSE #ENDIF

Permitem escolher qual o código que o compilador efetivamente observa.

```
#define DEBUG 1  
  
#if DEBUG>0  
    // do something  
#endif
```



# PRÉ-PROCESSADOR, DIRETIVAS E INCLUSÃO CONDICIONAL

O **pré-processador** é um programa especial que está previsto na norma do C.

Usado automaticamente pelo compilador de C para efetuar algumas transformações simples do texto do programa, antes deste ser compilado.

Todas as diretivas começam pelo símbolo '#'.

# COISAS QUE SE PODEM ESCREVER NOS PROGRAMAS

Definir **entidades**, e.g.

funções, variáveis, constantes, tipos, etc.

Escrever **expressões**, e.g.

fórmula matemática que envolva a  
velocidade de propagação do som.

Escrever **instruções**, e.g.

chamar a função printf para  
apresentar no ecrã um certo texto.

## DEFINIÇÕES

Uma **definição** dá origem uma nova **entidade**.  
Funções, variáveis, parâmetros de funções,  
constantes, tipos;

## EXPRESSÕES

Uma **expressão** é um pedaço de código que pode  
ser **avaliado** para produzir um **resultado**.

## INSTRUÇÕES

Uma **instrução** (ou **comando**) é um pedaço de  
código que executa **ações**.

# INSTRUÇÃO COMPOSTA (BLOCO)

É possível criar uma **instrução composta** a partir de duas ou mais instruções, usando chavetas {}.

```
int fatorial(int n) {  
    int resultado = 1;  
    do {  
        resultado = resultado * n;  
        n = n - 1;  
    }  
    while(n > 1);  
    return resultado;  
}
```

# INSTRUÇÕES DE CONTROLO

São instruções que exprimem a **tomada de decisões** ou a **execução de ciclos**.

Controlam o fluxo do programa.

Exemplos:

- for;
- if, else;
- while;
- do, while;

Identifique as  
definições,  
expressões  
e instruções.

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

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

```
1 double hipotenusa(double c1, double c2) {
```

```
2     return sqrt(c1 * c1 + c2 * c2);
```

```
3 }
```

```
4
```

```
5 int main()
```

```
6 {
```

```
7     double c1, c2;
```

```
8     printf("Introduza os valores dos catetos:");
```

```
9     scanf("%lf %lf", &c1, &c2);
```

```
10    printf("A hipotenusa = %lf\n.", hipotenusa(c1, c2));
```

```
11
```

```
12    return 0;
```

```
13 }
```

## ATIVIDADES QUE OCORREM DURANTE A EXECUÇÃO

- **Avaliação de Expressões** - A avaliação de uma expressão é um processo gradual que conduz à produção de um resultado.

## ATIVIDADES QUE OCORREM DURANTE A EXECUÇÃO

- **Avaliação de Expressões** - A avaliação de uma expressão é um processo gradual que conduz à produção de um resultado.
- **Execução de Ações** - Há três tipos de ações que podem ser executadas:
  - Modificação do valor de variáveis;
  - Leituras e escritas.
  - Controlo do fluxo de execução por meio das instruções **if**, **for**, etc.



## ATIVIDADES QUE OCORREM DURANTE A EXECUÇÃO

- **Avaliação de Expressões** - A avaliação de uma expressão é um processo gradual que conduz à produção de um resultado.
- **Execução de Ações** - Há três tipos de ações que podem ser executadas:
  - Modificação do valor de variáveis;
  - Leituras e escritas.
  - Controlo do fluxo de execução por meio das instruções **if**, **for**, etc.
- E as **Definições???**

# ALTERAR O VALOR DUMA VARIÁVEL: LEITURAS E ATRIBUIÇÕES

O valor de uma variável pode ser **alterado** em qualquer momento, de diversas formas.

```
int a = 5;  
int b = 6;  
int c;  
scanf ("%d", &a) ;
```

```
c = 13;
```

= é o operador de atribuição.

Mostra o que sabes!  
01 – O que é a programação?

<https://forms.gle/X8SAHVzM4pTFR77d6>



# OPERADORES DE ATRIBUIÇÃO

Outros operadores que realizam atribuição:

**`+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `&=` `^=` `|=`**

Estes operadores constituem simples abreviaturas, que combinam o operador básico de atribuição `=` com outras operações

`x += 2`

é abreviatura de

`x = x + 2`

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

```
int main()  
{  
    int n=2;  
    printf("\n x 2 = %d", n*2);  
    printf("\n = %d", n);  
    return 0;  
}
```

**O que será escrito  
na consola por este  
programa?**

# OPERADORES DE ATRIBUIÇÃO

Outros operadores que realizam atribuição:

`+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `&=` `^=` `|=`

Estes operadores constituem simples abreviaturas, que combinam o operador básico de atribuição `=` com outras operações

`x += 2`

é abreviatura de

`x = x + 2`

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

```
int main()
{
    int n=2;
    printf("\n x 2 = %d", n*2);
    printf("\n = %d", n);
    return 0;
}
```



```
4
2
```

# OPERADORES DE ATRIBUIÇÃO

Outros operadores que realizam atribuição:

`+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `&=` `^=` `|=`

Estes operadores constituem simples abreviaturas, que combinam o operador básico de atribuição `=` com outras operações

`x += 2`

é abreviatura de

`x = x + 2`

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

```
int main()
```

```
{
```

```
    int n=2;
```

```
    printf("\n x 2 = %d", n*=2);
```

```
    printf("\n = %d", n);
```

```
    return 0;
```

```
}
```

E agora?

# OPERADORES DE ATRIBUIÇÃO

Outros operadores que realizam atribuição:

`+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `&=` `^=` `|=`

Estes operadores constituem simples abreviaturas, que combinam o operador básico de atribuição `=` com outras operações

`x += 2`

é abreviatura de

`x = x + 2`

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

```
int main()
{
    int n=2;
    printf("\n x 2 = %d", n*=2);
    printf("\n = %d", n);
    return 0;
}
```

4

4

# INICIALIZAÇÃO DE VARIÁVEIS

Uma variável é definida e inicializada no ponto da definição, assim:

```
int x = 5;
```

O seu valor fica **definido**.

Também podemos inicializar uma variável usando a operação de leitura scanf e a operação de atribuição.



## VARIÁVEIS COM VALOR INDEFINIDO

Uma variável pode ser definida mas não inicializada no ponto da definição, assim:

```
int x;
```

As variáveis globais são implicitamente inicializadas a zero;

As variáveis locais não inicializadas têm valor **indefinido** - o compilador pode “fazer o que quiser”.

# VARIÁVEIS LOCAIS E VARIÁVEIS GLOBAIS

Uma **variável local** é uma variável definida dentro duma função.

Uma **variável global** é uma variável definida fora de qualquer função.

```
#include <stdio.h>

int x = 5;

void f(int a) { x = x + a; }

int main(void)
{
    f(4);
    printf("%d\n", x);
    return 0;
}
```

# OPERADORES DE INCREMENTO E DECREMENTO


Para incrementar e decrementar variáveis podemos também usar 2 operadores específicos

++      --

var++      pós-incremento

++var      pré-incremento

```
int main()  
{  
    int a = 1, b, c;  
    b = a++, c = ++a;  
    printf("a=%d, b=%d, c=%d, \n", a, b, c);  
}
```



# OPERADORES DE INCREMENTO E DECREMENTO

Para incrementar e decrementar variáveis podemos também usar 2 operadores específicos

++      --

var++      pós-incremento

++var      pré-incremento

```
int main()
{
    int a = 1, b, c;
    b = a++, c = ++a;
    printf("a=%d, b=%d, c=%d, \n", a, b, c);
}
```

→ a=3, b=1, c=3

→ ?

# CONSTANTES

Em C, chama-se **constante** a um valor fixo ao qual tenha sido dado um nome.

As constantes são definidas globalmente, usando a diretiva `#define` do pré-processador do C.

```
#define PI 3.1415926
#define PI_2 (PI / 2)
#define E 2.7182818284590452354
#define GRAVIDADE 9.8
#define SEGUNDOS_NUM_MINUTO 60
#define MINUTOS_NUMA_HORA 60
#define SEGUNDOS_NUMA_HORA (MINUTOS_NUMA_HORA * SEGUNDOS_NUM_MINUTO)
#define CAPACIDADE_TABELA 256|
```

Escreva um programa que calcule a área dum círculo, a partir do seu raio.

raio.c

```
1  #include <stdio.h>
2
3  #define PI 3.14159265358979323846
4
5  double area_circulo(double raio)
6  {
7      return raio * raio * PI;
8  }
9
10 int main(void)
11 {
12     double r;
13     printf("Introduza a medida do raio do círculo: ");
14     scanf("%lf", &r);
15     printf("Area do círculo = %lf\n", area_circulo(r));
16     return 0;
17 }
```

# TIPOS INTEIROS

Em C, existem diversas variedades de tipos inteiros. Os mais usados são:

- **int** - representar valores inteiros.
- **char** - representar caracteres (através dum código numérico).
- **bool** - representar os valores lógicos, false e true (usando internamente os valores 0 e 1).

Utilização em **printf** ou **scanf**:

- `%d` – inteiros;
- `%c` – caracteres;

**Nota:** para usar booleanos, é preciso usar a biblioteca para o efeito.

```
#include <stdbool.h>
```

# TIPOS INTEIROS - LITERAIS

Um **literal** representa um valor concreto numa linguagem de programação:

- literais inteiros: 0, 1, 21056, -123;
- literais char: '\n', '!', 'A', 'Z', 'a', 'z', '\_';

## **Literais char:**

Número inteiro que corresponde ao código interno desse carácter.



# TIPOS INTEIROS - LITERAIS

Um **literal** representa um valor concreto numa linguagem de programação:

- literais inteiros: 0, 1, 21056, -123;
- literais char: '\n', '!', 'A', 'Z', 'a', 'z', '\_';

## Literais char

Número inteiro que corresponde ao código interno desse carácter.

0	NULL	10	LF	20	DC4	30	RS	40	(	50	2	60	<	70	F	80	P	90	Z	100	d	110	n	120	x
1	SOH	11	VT	21	NAK	31	US	41	)	51	3	61	=	71	G	81	Q	91	[	101	e	111	o	121	y
2	STX	12	FF	22	SYN	32		42	*	52	4	62	>	72	H	82	R	92	\	102	f	112	p	122	z
3	ETX	13	CR	23	ETB	33	!	43	+	53	5	63	?	73	I	83	S	93	]	103	g	113	q	123	{
4	EOT	14	SO	24	CAN	34	"	44	,	54	6	64	@	74	J	84	T	94	^	104	h	114	r	124	
5	ENQ	15	SI	25	EM	35	#	45	-	55	7	65	A	75	K	85	U	95	_	105	i	115	s	125	}
6	ACK	16	DLE	26	SUB	36	\$	46	.	56	8	66	B	76	L	86	V	96	`	106	j	116	t	126	~
7	BELL	17	DC1	27	ESC	37	%	47	/	57	9	67	C	77	M	87	W	97	a	107	k	117	u	127	DEL
8	BS	18	DC2	28	FS	38	&	48	0	58	:	68	D	78	N	88	X	98	b	108	l	118	v		
9	TAB	19	DC3	29	GS	39	'	49	1	59	;	69	E	79	O	89	Y	99	c	109	m	119	w		

# TIPOS INTEIROS - LITERAIS

Esta representação de caracteres é útil quando é preciso fazer contas com caracteres.

```
⊕ char char_seguinte(char ch) // Determina qual o carácter que se segue a ch
{
    return ch + 1;
}

⊕ char para_maiusculas(char ch) // Converte letra minúscula na letra maiúscula correspondente
// Assume-se que ch é uma letra minúscula
{
    return ch - 'a' + 'A';
}

⊕ bool eh_algarismo(char c) // Testa se um carácter é um algarismo
{
    return '0' <= c && c <= '9';
}
```

## TIPOS INTEIROS - OPERAÇÕES

- Em C, há 5 operações aritméticas sobre inteiros:

+          -          \*          /

% (resto da divisão)

## TIPOS INTEIROS - TRANSBORDO

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

```
int main()
{
    int a = INT_MAX; //2147483647
    int b = a+1;
    printf("a=%d, b=%d\n", a, b);
}
```

→ a=2147483647, b=-2147483648

# TODOS OS TIPOS INTEIROS DA LINGUAGEM C

Nome habitual	Nome completo	Bytes	Valor mínimo	Valor máximo
bool	bool	1	0 ( <i>false</i> )	1 ( <i>true</i> )
signed char	signed char	1	-128	+127
unsigned char	unsigned char	1	0	255
char	<i>um dos dois atrás</i>	-	0 ( <i>na prática</i> )	127 ( <i>na prática</i> )
short	signed short int	2	-32,768	+32,767
unsigned short	unsigned short int	2	0	65,535
int	signed int	4	-2,147,483,648	+2,147,483,647
unsigned	unsigned int	4	0	4,294,967,295
long	signed long int	4	-2,147,483,648	+2,147,483,647
unsigned long	unsigned long int	4	0	4,294,967,295
long long	signed long long int	8	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807
unsigned long long	unsigned long long int	8	0	18,446,744,073,709,551,615

# ***CARACTERES - CHAR***

O tipo **char** permite armazenar **um único caracter**.

Declaração:

- `char var1, ch, var2;`

Inicialização:

- `char var1 = 'A', ch = 'F', var2 = 'A';`
- `scanf("%c", &var1);`

Exemplos de alguns caracteres especiais:

- `\b` - backspace
- `\n` – new line
- `\t` – tabulação horizontal
- `\\` - caracter `'\'`
- `\"` – caracter `"`

## ***CARACTERES - CHAR***

- **getchar()** e **scanf()**:

```
var1 = getchar();
```

- **printf()** e **putchar()**:

```
putchar( 'A' );
```

### **Exercício**

Escreva um programa que solicite ao utilizador dois caracteres, separadamente, utilizando a função `scanf()`.

# CARACTERES E INTEIROS

Os caracteres não são mais que valores inteiros guardados num único Byte.

```
char ch;  
  
ch = 'A'; //Formato tradicional  
ch = 65; //Caracter cujo código ASCII é 65
```

```
1 #include <stdio.h>  
2  
3 int main(void){  
4     char ch;  
5  
6     printf("Introduza um Caracter: ");  
7     scanf("%c",&ch);  
8     printf("O caracter '%c' tem o ASCII nº %d \n",ch,ch);  
9     return 0;  
10 }
```

## ***CARACTERES E INTEIROS***

Apesar de se poder realizar conversão de inteiros para caracteres e vice-versa, a leitura de inteiros com formato %c ou a leitura de caracteres com o formato %d, leva a resultados inesperados e perigosos, não devendo por isso ser realizada.

Exemplo na página 66 do livro do Luís Damas



# TIPOS REAIS

Em C o tipo real mais usado é o `float` e o `double`;

Exemplos de literais reais:

`3.14159`, `12.3e56`, `-1e-2`, `12.0`.

Utilização em `printf` ou `scanf`:

- `%f`
- `%e` (notação científica)
- `%lf` (double)

## TIPOS REAIS - OPERAÇÕES

Em C, há 4 **operações aritméticas** sobre reais:

+

-

\*

/

# TIPOS REAIS - OPERAÇÕES

Operações disponíveis na biblioteca <math.h>

Função	Descrição
cos	coseno
sin	seno
tan	tangente
acos	arco-coseno
asin	arco-seno
atan	arco-tangente
cosh	coseno hiperbólico
sinh	seno hiperbólico
tanh	tangente hiperbólica
exp	exponencial
log	logaritmo natural
log10	logaritmo de base 10
pow	power (x elevado a y)
sqrt	raiz quadrada
ceil	teto (o inteiro mais pequeno que não é inferior ao argumento)
floor	chão (o inteiro maior que não é superior ao argumento)
fabs	valor absoluto
fmod	resto da divisão

# TIPOS REAIS - TRANSBORDO

Uma variável real tem uma capacidade limitada. Quando se tenta fazer uma variável real exceder os limites, nessa variável fica um de três valores especiais:

- Inf
- -inf
- nan (not a number)

## TODOS OS TIPOS REAIS DA LINGUAGEM C

Nome habitual	Bytes	Dígitos de precisão
float	4	6
double	8	16
long double	10	20

# TIPOS COMPLEXOS

- Também existem tipos para representar números complexos:
  - float complex
  - double complex
  - long double complex.

```
complex.c
1 #include <stdio.h>
2 #include <complex.h>
3
4 int main(void)
5 {
6     // Cria complexo, com parte real e imaginária
7     complex z = 2.0 + 2.0*I;
8     // Repare nas funções de acesso às duas partes do complexo z
9     printf("z = %lf + %lfI\n", creal(z), cimag(z));
10    return 0;
11 }
```

# TIPOS VOID

O tipo **void** representa um **conjunto vazio de valores**.

Não é possível definir variáveis do tipo void.

Mas o tipo void pode aparecer no cabeçalho da definição de funções:

- Na zona dos argumentos
- Na zona do tipo do resultado que retorna

```
void mostrar_algo(void) {  
    printf("função só imprime algo\n");  
}
```

# EXPRESSÕES

## Operadores

```
aritméticos:  +  -  *  /  %
lógicos:      !  &&  ||
relacionais:  <  >  <=  >=  ==  !=
bits:         >>  <<  &  ^  |  ~
atribuição:   =  +=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=
condicional   ?:
cast:         (type)
inc/dec:      expr++  ++expr  expr--  --expr
sequenciação: ,
sizeof:       sizeof
apontadores:  *  &  ->  []
field:        .          (para acesso a campos de registos e uniões)
agrupamento: (expr)
```

# AVALIAÇÃO DE EXPRESSÕES

## Ordem de avaliação

O compilador de C tem a liberdade de reorganizar as expressões por forma a otimizar a eficiência da sua avaliação.

## Exemplos:

```
Func1() + func2()  
i = i++
```

## Pontos de sequenciação

Pontos dentro das expressões que garantem que as ações das expressões anteriores já foram completamente concretizadas:

,	(operador de sequenciação)
&&	(e lógico)
	(ou lógico)

# CONVERSÕES AUTOMÁTICAS ENTRE TIPOS NUMÉRICOS

Tipos numéricos podem ser livremente misturados em expressões, e.g.

$$23 + 4.56$$

As regras gerais de conversão automática de tipo são relativamente simples:

1. **char** e **short** são convertidos em **int**; **float** são convertidos **double**.
2. Para cada operador binário com operandos de tipo diferente, **o operando de tipo menos importante é convertido para o tipo mais importante**, antes de se aplicar a operação binária.
3. Nas atribuições  $v = \text{exp}$ , **o tipo do valor da expressão da direita é convertido num valor do tipo da esquerda** antes de se fazer a atribuição. Muitas vezes isso implica fazer uma despromoção de tipo.



# CONVERSÕES AUTOMÁTICAS ENTRE TIPOS NUMÉRICOS

Para efeitos de promoções e despromoções, a hierarquia dos tipos numéricos é a seguinte:

```
long double          // mais importante
double
unsigned long long int
long long int
unsigned long int
long int
unsigned int
int                  // menos importante
```

```
printf("%d\n", 5+2.0);
printf("%f\n", 5+2.0);
printf("%c\n", 'a');
printf("%d\n", 'a');
printf("%lf\n", 200.0 * 'a');
```

O que irá aparecer na consola?

# CONVERSÕES AUTOMÁTICAS ENTRE TIPOS NUMÉRICOS

Para efeitos de promoções e despromoções, a hierarquia dos tipos numéricos é a seguinte:

```
long double          // mais importante
double
unsigned long long int
long long int
unsigned long int
long int
unsigned int
int                  // menos importante
```

```
printf("%d\n", 5+2.0);
printf("%f\n", 5+2.0);
printf("%c\n", 'a');
printf("%d\n", 'a');
printf("%lf\n", 200.0 * 'a');
```

```
108106424
7.000000
a
97
19400.000000
```

## *casts* - OPERADORES DE CONVERSÃO EXPLÍCITA DE TIPO

Em algumas situações raras pode ser necessário usar um cast, ou seja, um **operador conversão explícita**.

```
printf("%d\n", 1/3);  
printf("%lf\n", 1/3);  
printf("%lf\n", 1/3.0);  
printf("%lf\n", 1.0/3);  
printf("%lf\n", 1/(double)3);
```

O que irá aparecer na consola?

Outros operadores de cast disponíveis:

- (int)
- (unsigned)
- (long long)
- etc...

## *casts* - OPERADORES DE CONVERSÃO EXPLÍCITA DE TIPO

Em algumas situações raras pode ser necessário usar um cast, ou seja, um **operador conversão explícita**.

```
printf("%d\n", 1/3);  
printf("%lf\n", 1/3);  
printf("%lf\n", 1/3.0);  
printf("%lf\n", 1.0/3);  
printf("%lf\n", 1/(double)3);
```

```
0  
0.000000  
0.333333  
0.333333  
0.333333
```

Outros operadores de cast disponíveis:

- (int)
- (unsigned)
- (long long)
- etc...

# ***FORMATOS DE LEITURA E ESCRITA (RESUMO)***

página 69 – Livro Luís Damas

## **Texto**

`char - %c;`

## **Números naturais**

<code>int</code>	<code>%d</code>
<code>short int</code>	<code>%hd</code>
<code>long int</code>	<code>%ld</code>
<code>unsigned short int</code>	<code>%hu</code>
<code>unsigned int</code>	<code>%u</code>
<code>unsigned long int</code>	<code>%lu</code>

## **Números reais**

<code>float</code>	<code>%f, %e</code>
<code>double</code>	<code>%lf, %E</code>

Programas  
para analizar

Escreva um programa que calcule o Teorema de Pitágoras.

Escreva um programa que calcule o Teorema de Pitágoras.

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

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

```
double hipotenusa(double c1, double c2){
```

```
    return sqrt(c1 * c1 + c2 * c2);
```

```
}
```

```
int main()
```

```
{
```

```
    double c1, c2;
```

```
    printf("Introduza os valores dos catetos:");
```

```
    scanf("%lf %lf", &c1, &c2);
```

```
    printf("A hipotenusa = %lf\n.", hipotenusa(c1, c2));
```

```
    return 0;
```

```
}
```



Escreva um programa que determine o número de dias dum ano qualquer.

Escreva um programa que determine o número de dias dum ano qualquer.

```
#include <stdio.h>

//sao bisextos os multiplos de 4, excepto se for multiplo de 100 mas nao de 400
int bisexto(int ano){
    if ( (ano % 4 == 0 && ano % 100 != 0) || (ano % 400 == 0) ) return 1;
    else return 0;
}

int diasnoano(int ano){
    return 365 + bisexto(ano);
}

int main()
{
    int n;

    printf("Introduza um ano:");
    scanf("%d", &n);
    printf("%d tem %d dias\n.", n, diasnoano(n));

    return 0;
}
```

Escreva um programa que calcule o fatorial de um número inteiro não-negativo, usando da forma direta a definição recursiva , nossa conhecida da Matemática.

$$n! = n \times (n - 1)!$$

Escreva um programa que calcule o fatorial de um número inteiro não-negativo, usando da forma direta a definição recursiva , nossa conhecida da Matemática.

$$n! = n \times (n - 1)!$$

```
#include <stdio.h>

int fatorial(int n){
    return (n == 0) ? 1 : n * fatorial(n-1);
}

int main()
{
    int val;
    printf("Introduza um valor inteiro:");
    scanf("%d", &val);
    printf("fatorial de %d e %d\n.", val, fatorial(val));

    return 0;
}
```

Outra forma de calcular o fatorial.

```
#include <stdio.h>

int fatorial(int n)
{
    int i, fact = 1;
    for( i = 1 ; i <= n ; i = i + 1 )
        fact = fact * i;
    return fact;
}

int main(void)
{
    int n;
    printf("Introduza um inteiro nao negativo: ");
    scanf("%d", &n);
    printf("fatorial(%d) = %d\n", n, fatorial(n));
    return 0;
}
```

Escreva um programa que dado um número inteiro positivo, apresente de forma sequencial todos os números desde zero até o seu valor.

Escreva um programa que dado um número inteiro positivo, apresente de forma sequencial todos os números desde zero até o seu valor.

```
#include <stdio.h>

void apresenta_valor(int valor){
    int i;
    for(i=0; i<valor; i++)
        printf("%d\n", i);
}

int main()
{
    int valor;
    printf("indique um valor inteiro positivo:\n");
    scanf("%d", &valor);
    apresenta_valor(valor);

    return 0;
}
```

Escreva um programa que indique se o valor real introduzido é positivo ou negativo.



Escreva um programa que indique se o valor real introduzido é positivo ou negativo.

```
real.c ✖
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  bool testaValor(int x){
5      if (x>0)
6          return true;
7
8      return false;
9  }
10
11  int main(void){
12      int valor;
13
14      printf("Introduza o valor real a testar:\n");
15      scanf("%d", &valor);
16      if (testaValor(valor))
17          printf("O valor é positivo!\n");
18      else
19          printf("O valor é negativo!\n");
20  }
```

# Programa para arredondar valores reais positivos

# Programa para arredondar valores reais positivos

```
round.c ✖
1 #include <stdio.h>
2 #include <math.h>
3
4 double arredondamento(double x)
5 {
6     if (x >= 0)
7         return floor(x + 0.5);
8     else
9         return -floor(fabs(x) + 0.5);
10    // return ceil(x - 0.5); // daria o mesmo resultado
11 }
12
13 int main(void)
14 {
15     double v;
16     printf("Qual o valor a arredondar? ");
17     scanf("%lf", &v);
18     printf("%lf\n", arredondamento(v));
19     return 0;
20 }
```

**EXTRA**

# EXPRESSÕES

## Operador de sequenciação ,

- é um operador binário, que se usa assim:
  - Expressão1, Expressão2
- Esta expressão composta é avaliada assim:
  1. avalia-se a subexpressão da esquerda;
  2. avalia-se a subexpressão da direita;
  3. o valor da expressão composta é o valor da subexpressão da direita.

```
int a=1, b=2;  
int i = (a += 2, a + b);
```



a=3, b=2, i=5