

# Linguagem C

Programação em Sistemas Computacionais

João Pedro Patriarca ([jpatri@cc.isel.ipl.pt](mailto:jpatri@cc.isel.ipl.pt), [joao.patriarca@isel.pt](mailto:joao.patriarca@isel.pt)), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

# Sumário

---

- Aspectos básicos da linguagem
- Tipos inteiros
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Sumário

---

- Aspectos básicos da linguagem
- Tipos inteiros
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Aspetos da linguagem C

---

- Programação de sistemas
- Desenhado para estar “perto” do hardware
- Acesso direto à memória
- Portabilidade apenas ao nível do código fonte
  - Um executável gerado para arquiteturas a 64 bits não corre em arquiteturas a 32 bits
  - Um executável gerado para plataforma Windows não corre numa plataforma Linux
- Linguagem tipificada
- Linguagem procedimental
- Gestão de memória feita explicitamente (não existe *garbage collection*)
- Detecção de erros explícita
  - Não são geradas exceções (ex: referências a NULL, indexação fora dos limites, ...)
- Mais linhas de código para a mesma funcionalidade quando comparado com outras linguagens de programação (e.g. Kotlin, Java, ...)

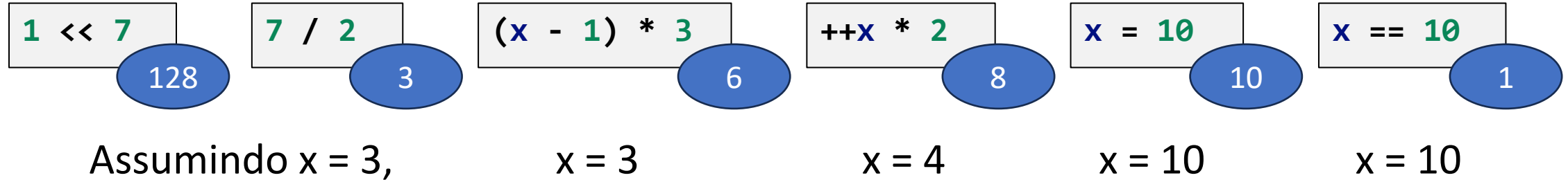
# Exemplo de um programa escrito em C

```
#include <stdio.h>
/* Count the number of words in a file.
 * Use Ctrl^D to produce EOF from command line.
 * Use ./words < file to redirect stdin to file. */
int main() {
    int nextChar;    // char readed
    int count = 0;   // words counter
    int inWord = 0;  // current state
    while (( nextChar = getchar() ) != EOF) {
        if (nextChar == '\n' || nextChar == ' ')
            inWord = 0;
        else if (! inWord) {
            inWord = 1;
            count += 1;
        }
    }
    printf("Words count=%d\n", count);
    return 0;
}
```

# Expressões, instruções e blocos

Bib: (A), cap. 2

- Uma expressão produz um valor



- Instrução é uma expressão terminada pelo caracter ‘;’

```
x = (x - 1) * 3;
```

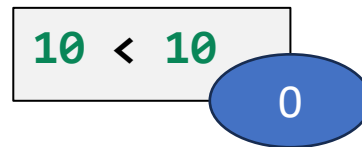
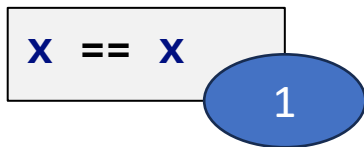
```
x = 10; x = x == 10;
```

- Bloco corresponde a um conjunto de uma ou mais instruções envolvidas pelos caracteres ‘{’ e ‘}’
  - Corpo de funções, bloco *if*, *while*, *for*, ... (ver exemplo anterior)

# Instruções de controlo e valores lógicos

Bib: (A), cap. 3

- Instruções de decisão: *if-else*
- Instruções de ciclo: *do-while*, *while*, *for*
- Na linguagem C não existe o tipo *boolean*
  - Uma expressão de controlo é verdadeira se o resultado da expressão for diferente de 0
  - O resultado da avaliação das operações relacionais é 1 ou 0



```
// Produz em stdout: .....  
int x = 5;  
while (x--) {  
    putchar('.');  
}
```

- Identificadores
  - Sequência de letras, de dígitos e de símbolos `_` (travessão em baixo)
  - Não podem começar por dígito
  - São *case-sensitive* mas não é boa prática
- Tipos básicos: *char*, *int*, *float*, *double*
  - Modificadores de sinal (*unsigned* e *signed* (por omissão)) aplicáveis aos tipos *char* e *int*
  - Modificadores de dimensão (*short* e *Long*) aplicáveis ao tipo *int*
- Constantes
  - *int*: `23`, `023` (octal), `0x23` (hexadecimal), `23L` (long), `23U` (unsigned)
  - *float* ou *double*: `23.5`, `23.5F`, `23.`, `23.F`, `12.3e-4`, `12.3e10F`
  - *char*: `'m'`, `'M'`, `'\n'`, `'\t'`, `'\''`, `'\\'`, `'\0'`, `'\48'`, `'\x30'`
  - String: `"123"`, `"abc"`, `"aspas=\""`, `"Linha\n"`, `"backspace=\b"`



# Variáveis/declarações

Bib: (A), cap. 2

- Uma variável tem um nome e um tipo

```
char c;      // Declaração de variável não iniciada  
int i = 10;  // Declaração de variável iniciada
```

- O tipo da variável define o significado e a dimensão do valor que consegue armazenar
- A visibilidade de uma variável corresponde ao âmbito (*scope*) e depende de onde está declarada:
  - Local ao bloco (funções e instruções de controlo)
  - Global (declarada fora de qualquer função)
- O tempo vida de uma variável (desde a reserva à libertação de memória) está relacionada com o tipo de alocação:
  - Alocação automática (sujeito ao bloco)
  - Alocação estática (sujeito ao programa)
  - Alocação dinâmica (sujeito ao programador)
- Uma variável pode ser marcada como constante com o qualificador *const*

```
const double PI = 3.1415926535;
```

# Sumário

---

- Aspectos básicos da linguagem
- **Tipos inteiros**
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Tipo *char*

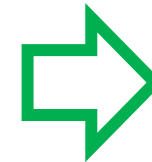
Bib: (A), cap. 2

Tipo	Utilização	Dimensão	Intervalo de valores
<i>signed char</i>	Valores pequenos Código de um caracter	8 bits (1 byte)	-128 a +127
<i>unsigned char</i>			0 a 255
<i>char</i>			Definido na implementação

```
void f6_char_type() {
    char up='A', lo;
    for(lo = up - 'A' + 'a' ; up <= 'Z' ; ++up, ++lo)
        printf("%c-%d %c-%d\n", up, up, lo, lo);

    signed char sc= -1;
    unsigned char uc= -1;
    char c = -1;
    printf("%d, %d, %c, %d\n", sc, uc, c, c);

    printf("sizeof(char): %d\n", (int)sizeof(c));
}
```



A-65	a-97
B-66	b-98
C-67	c-99
D-68	d-100
...	
W-87	w-119
X-88	x-120
Y-89	y-121
Z-90	z-122
-1,	255, 0, -1
sizeof(char): 1	

# Tipo *int*

Bib: (A), cap. 2

Tipo	Utilização	Dimensão típica (64 bits)	Intervalo de valores
<i>signed int</i> ou <i>int</i>	Valores inteiros	32 bits (4 bytes)	-2147483648..+2147483647
<i>unsigned int</i>			0..4294967295
<i>long int</i> ou <i>long</i>	Valores inteiros grandes	32 ou <u>64</u> bits (4 ou <u>8</u> bytes)	-9223372036854775808.. +9223372036854775807
<i>unsigned long</i>			0..18446744073709551615
<i>short int</i> ou <i>short</i>	Valores inteiros pequenos	16 bits (2 bytes)	-32768..+32767
<i>unsigned short</i>			0..65535

Standard garante apenas:  $\text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$

# Tipo *int* (exemplo)

```
void f7_int_type() {
    short min_s, max_s; int min_i, max_i;
    long min_l, max_l; unsigned short us = ~0;
    unsigned int ui = ~0; unsigned long ul = ~0;
    max_s = us >> 1;    // Logical shift right
    min_s = us ^ max_s; // bit wise xor
    max_i = ui >> 1;    // Logical shift right
    min_i = ui ^ max_i; // bit wise xor
    max_l = ul >> 1;    // Logical shift right
    min_l = ul ^ max_l; // bit wise xor
    printf("min..max unsigned\n"
        "\tunsigned short: 0..%d\n"
        "\tunsigned int:   0..%u\n"
        "\tunsigned long:   0..%lu\n\n", us, ui, ul);
    printf("min..max signed\n"
        "\tshort: %d..+%d\n\tint:   %d..+%d\n"
        "\tlong:  %ld..+%ld\n\n",
        min_s, max_s, min_i, max_i, min_l, max_l);
    printf("sizeof(short) = %d\n"
        "sizeof(int) = %d\nsizeof(long)= %d\n",
        (int)sizeof(short), (int)sizeof(int),
        (int)sizeof(long));
}
```



```
min_max unsigned
short: 0..65535
int:   0..4294967295
long:  0..18446744073709551615

min..max signed
short: -32768..+32767
int:   -2147483648..
        +2147483647
long:  -9223372036854775808..
        +9223372036854775807

sizeof(short):  2
sizeof(int):    4
sizeof(long):   8
```

# Tipo *float* e *double*

Bib: (A), cap. 2

Tipo	Utilização	Dimensão (norma IEEE754)	Expoente (bits)	Mantissa (bits)	Intervalo de valores
<i>float</i>	Valores reais pequenos	32 bits (4 bytes)	8	23	$\pm 1.40 \times 10^{-45}..$ $\pm 3.40 \times 10^{+38}$
<i>double</i>	Valores reais grandes	64 bits (8 bytes)	11	52	$\pm 4.94 \times 10^{-324}..$ $\pm 1.76 \times 10^{+308}$

- Codificação de acordo com a norma IEEE754

- Conversão implícita (promoção)
  - Quando o tipo destino é um super conjunto do tipo origem
  - Verificada em tempo de compilação
  - Numa expressão envolvendo tipos diferentes, as operações são realizadas no tipo mais abrangente
- Conversão explícita (despromoção)
  - Quando o tipo destino é um subconjunto do tipo origem
  - O valor convertido pode perder representação (o valor convertido é truncado)
  - O programador precisa fazer *cast*

# Conversão entre tipos (exemplo)

```
void f8_type_conversions() {  
    unsigned char c1 = ~0;  
    unsigned int i1 = c1;  
    unsigned long l1 = i1;  
    unsigned long l2 = ~0;  
    unsigned int i2 = (unsigned int)l2;  
    unsigned char c2 = (unsigned char)i2;  
    printf("Promotions:\n\tc=%u, i=%u, l=%lu\n\tc=%d, i=%d, l=%ld\n"  
        "Demotions:\n\tul=%lu, ui=%u, uc=%u\n\tul=%ld, ui=%d, uc=%d\n",  
        c1, i1, l1, c1, i1, l1, l2, i2, c2, l2, i2, c2);  
}
```



```
Promotions:  
    c=255, i=255, l=255  
    c=255, i=255, l=255  
Demotions:  
    ul=18446744073709551615, ui=4294967295, uc=255  
    ul=-1, ui=-1, uc=255
```



# Sumário

---

- Aspectos básicos da linguagem
- Tipos inteiros
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Operadores

Bib: (A), cap. 2

- Aritméticos:
  - Quatro operadores entre inteiros e reais:  $+$ ,  $-$ ,  $*$  e  $/$
  - Resto de divisão inteira:  $\%$
  - Operadores unários:  $+$  e  $-$
  - Incremento e decremento (prefixo e sufixo):  $++$  e  $--$
- Relacionais:  $>$ ,  $>=$ ,  $<$ ,  $<=$ ,  $==$  e  $!=$
- Lógicos:  $\&\&$ ,  $||$  e o unário  $!$
- Bit a bit:  $\&$ ,  $/$ ,  $^$ ,  $<<$ ,  $>>$  e o unário  $\sim$
- Afetação:  $=$ 
  - Compostos:  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$ ,  $\&=$ ,  $|=$ ,  $\wedge=$ ,  $<<=$  e  $>>=$

## Precedência e associatividade

()	[]	->	.		→
!	~	++	--	+	←
-	*	&	(type)	sizeof	
*	/	%			→
+	-				→
<<	>>				→
<	<=	>	>=		→
==	!=				→
&					→
^					→
					→
&&					→
					→
?:					←
=	+=	-=	...		←
,					→

# Sumário

---


- Aspectos básicos da linguagem
- Tipos inteiros
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Função *printf*("", ...)

Bib: (A), cap. 7.2

- Escreve no *stdout* (*standard output stream*) uma *string* formatada
- 1º parâmetro: *string* de formatação
- 2º, ..., n parâmetros: parâmetros de acordo com a *string* de formatação
- Tipos de formatação:
  - `%c` – caracter correspondente ao código
  - `%d` – valor inteiro decimal com sinal
  - `%u` – valor inteiro decimal sem sinal
  - `%x` – valor inteiro hexadecimal
  - `%L[d|u|x]` – o prefixo l (L minúsculo) interpreta o valor como long
  - `%f` – valor real
  - `%s` – *string*
  - `%p` – valor de ponteiro (em hexadecimal)
- Outras funções:
  - `int getchar();` //Devolve um caracter do stdin
  - `int putchar( int ch );` // Escreve o caracter ch  
// no stdout

```
void f9_printf() {  
    int i = -100;  
    char * str = "abc";  
    printf("i: %%c=%c, %%u=%u, \n"  
          "      %%d=%d, %%x=%x, \n"  
          "      %%lx=%lx, \n"  
          "      %%f=%f\n"  
          "str: %%s=\"%s\" \n",  
          i, i, i, i, (long)i,  
          (double)i, str);  
}
```



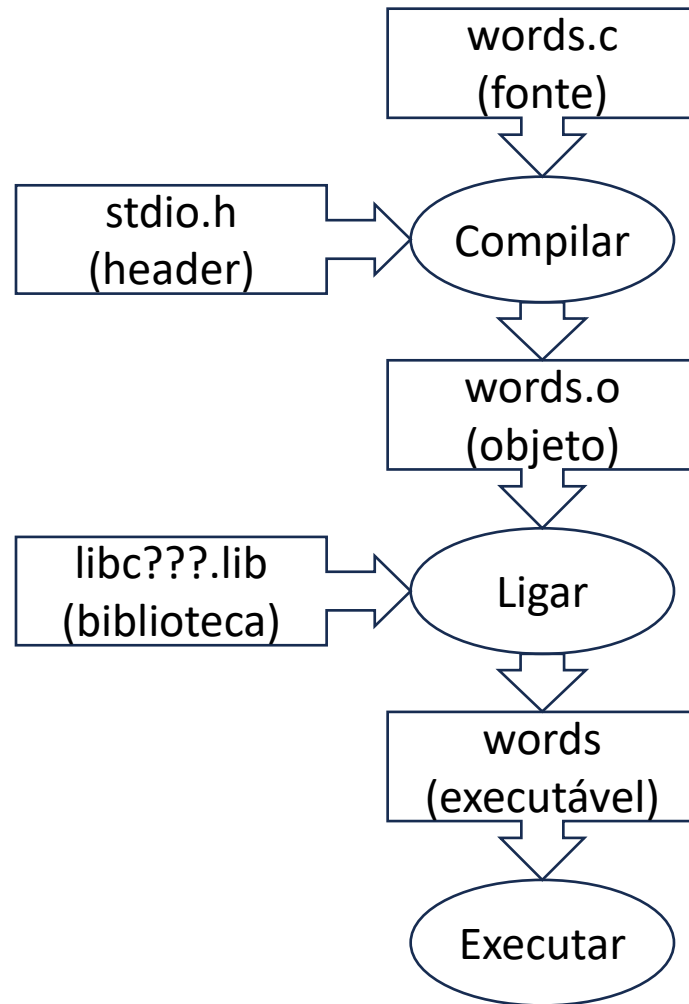
```
i: %c=␣, %u=4294967196,  
    %d=-100, %x=ffffff9c,  
    %lx=ffffffffffffff9c,  
    %f=-100.000000  
str: %s="abc"
```

# Sumário

---

- Aspectos básicos da linguagem
- Tipos inteiros
- Operadores
- Função *printf()*
- Comandos para gerar um executável

# Compilação, ligação e execução (Linux/GNU)



```
$ gcc -Wall -pedantic -g -c words.c
```

- Assinala erros de compilação: pré-processador; sintaxe; semânticos
  - Wall e -pedantic: ativa todos os *warnings*
  - g: gera informação de *debug*
  - c: interrompe o processo após compilação

```
$ gcc words.o -o words
```

- Assinala erros de ligação: ausência de símbolo ou repetição de símbolo
  - o: nome do ficheiro executável

```
$ ./words < text_file.txt
```

- Neste caso, o *stdin* é redirecionado para o ficheiro de texto *text\_file.txt*

# Exercícios

---

1. Apresentar no *stdout* o número de caracteres dígito presentes num ficheiro de texto
2. Apresentar no *stdout* o valor máximo com sinal e o valor mínimo com sinal do tipo *int*
3. Compactador/descompactador de uma data
  - Assume-se que o ano compactado codifica apenas um ano a partir do ano 2000 (inclusive)

*unsigned short date\_pack(int y, int m, int d);*

*int date\_unpack\_year(unsigned short date);*

*int date\_unpack\_month(unsigned short date);*

*int date\_unpack\_day(unsigned short date);*

- Formato da data compactada

