Tipos de Dados em C

Alexsandro Santos Soares
prof.asoares@gmail.com

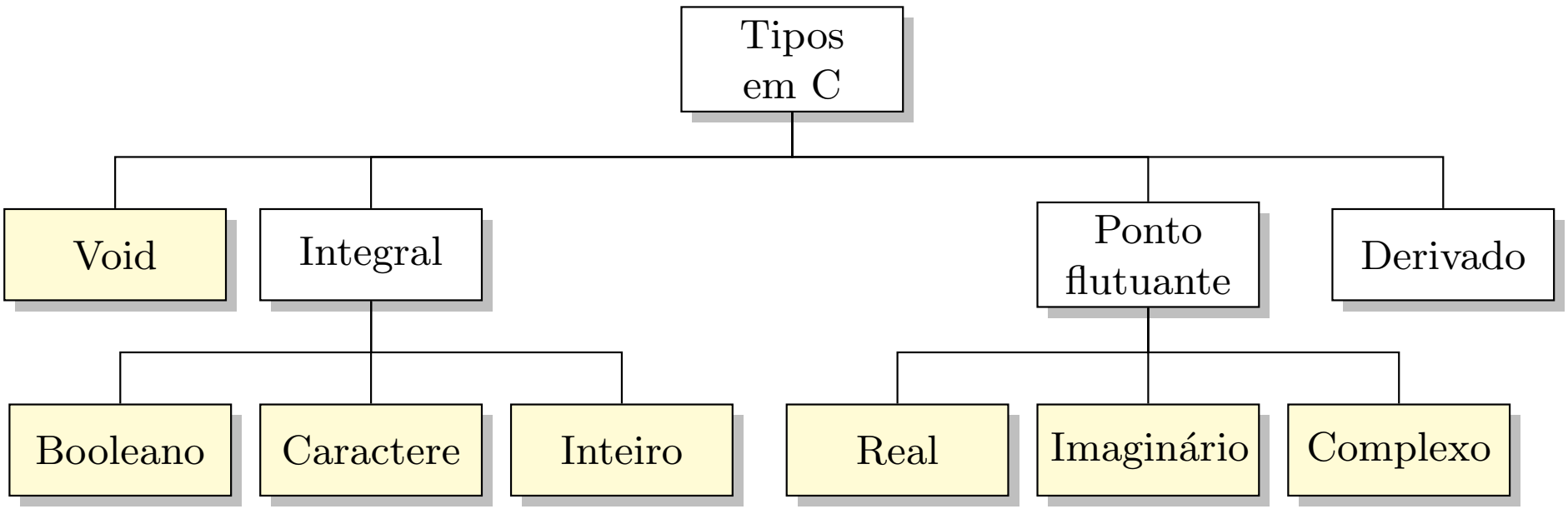
Universidade Federal de Uberlândia
Faculdade de Computação

Tipos

- Em C, o termo objeto se refere a um endereço na memória cujo conteúdo representa valores.
- Um tipo define um conjunto de valores e um conjunto de operações que podem ser aplicadas a esses valores.
- O tipo do objeto determina quanto espaço ele ocupará na memória e como seus possíveis valores serão codificados.
  - Como vimos, o mesmo padrão de bits pode designar números completamentamente diferentes tais como: um número em ponto flutuante, inteiro com ou sem sinal, etc.

Tipos

- A linguagem C define um conjunto de tipos que podem ser divididos em quatro categorias: void, integral, ponto flutuante e derivado.
- Esses tipos são mostrados na diagrama abaixo.
- Por enquanto nos concentraremos nos três primeiros.



Void

- O tipo void, representado pela palavra-chave `void`, não possui valor ou operações.
- Ele é usado, entre outras coisas, para indicar que uma função não tem argumentos, como vimos anteriormente na declaração da função `main`.
- Ele também pode ser usado para definir uma função que não possua nenhum valor de retorno.
- Outro uso é na definição de um ponteiro para um tipo de dados genérico.
- Estes dois últimos usos entenderemos melhor daqui a algumas aulas.

Tipos integrais

- A linguagem C possui três tipos integrais: booleanos, caracteres e inteiros.
- Tipos integrais não possuem partes fracionárias.
- Eles são números inteiros.

Booleanos

- A partir do padrão publicado em 1999, à linguagem C foi incorporado um tipo booleano.
- Um tipo booleano possui dois valores: `true` e `false`.
- Antes de 1999, C usava inteiros para representar valores booleanos:
  - Qualquer valor diferente de zero, seja ele positivo ou negativo, representava verdadeiro (*true*).
  - Zero era usado para representar falso (*false*).
- Por questões de compatibilidade, os inteiros ainda podem ser usados para representar valores booleanos.
  - Neste curso sempre usaremos o tipo booleano!
- Um tipo booleano é declarado com a palavra reservada `bool` e seus valores são armazenados na memória como 0 (`false`) e 1 (`true`).

Caractere

- Um caractere é qualquer valor que possa ser representado no alfabeto do computador, ou como é melhor conhecido, seu conjunto de caracteres.
  - Letras, dígitos, sinais e outros símbolos que possam ser digitados.
- Existem três tipos de caracteres em C11:

  `signed char` para a representação de caracteres com sinal.

  `unsigned char` para a representação de caracteres sem sinal ou de um byte na memória.

  `char` para a representação de caracteres.
- A biblioteca padrão do C11 também define os nomes `wchar_t`, `char16_t` e `char32_t` para representar caracteres longos.
- Nos próximos slides veremos a diferenças entre estes vários tipos.

ASCII

- Muitos computadores utilizam o alfabeto da American Standard Code for Information Interchange (ASCII).
- Muitos computadores usam 1 byte (8 bits) para armazenar o tipo de dados `char`.
  - Com 8 bits é possível representar 256 valores diferentes no conjunto de caracteres.
  - O ASCII somente utiliza metade desses valores possíveis.
- O tamanho de um `char` pode variar de uma máquina para outra, mas normalmente é 1 byte.
- No próximo slide apresentaremos uma parte da tabela ASCII

Tabela ASCII – parcial

Char	Dec	Oct	Hex
(nul)	0	000	00
(nl)	10	012	0a
(cr)	13	015	0d
(space)	32	040	20
0	48	060	30
1	49	061	31
2	50	062	32
3	51	063	33
4	52	064	34
5	53	065	35
6	54	066	36
7	55	067	37
8	56	070	38
9	57	071	39
A	65	101	41
B	66	102	42
C	67	103	43
D	68	104	44
E	69	105	45
F	70	106	46
G	71	107	47
H	72	110	48
I	73	111	49
J	74	112	4a
K	75	113	4b
L	76	114	4c
M	77	115	4d
N	78	116	4e

Char	Dec	Oct	Hex
O	79	117	4f
P	80	120	50
Q	81	121	51
R	82	122	52
S	83	123	53
T	84	124	54
U	85	125	55
V	86	126	56
W	87	127	57
X	88	130	58
Y	89	131	59
Z	90	132	5a
a	97	141	61
b	98	142	62
c	99	143	63
d	100	144	64
e	101	145	65
f	102	146	66
g	103	147	67
h	104	150	68
i	105	151	69
j	106	152	6a
k	107	153	6b
l	108	154	6c
m	109	155	6d
n	110	156	6e
o	111	157	6f
p	112	160	70

Char	Dec	Oct	Hex
q	113	161	71
r	114	162	72
s	115	163	73
t	116	164	74
u	117	165	75
v	118	166	76
w	119	167	77
x	120	170	78
y	121	171	79
z	122	172	7a
Ç	128	200	80
ü	129	201	81
é	130	202	82
â	131	203	83
ä	132	204	84
à	133	205	85
ç	135	207	87
ê	136	210	88
ë	137	211	89
®	169	251	a9
¬	170	252	aa
½	171	253	ab
¼	172	254	ac
+	197	305	c5
ã	198	306	c6
Ã	199	307	c7
¾	243	363	f3

Unicode

- O ASCII foi feito principalmente para tratar o léxico da língua inglesa.
  - Tentativas de estender o ASCII, chamado de ASCII estendido, tentaram usar os outros 128 valores disponíveis para caracteres de outras línguas.
  - O ASCII estendido nunca foi padronizado internacionalmente.
- Muitas línguas precisam de alfabetos diferentes daquele do Inglês.
- Para eliminar as limitações do ASCII a ISO e o Consórcio Unicode criaram um sistema de codificação universal para representar um conjunto de caracteres mais amplo: o Unicode.
- A versão moderna do Unicode codifica um caractere com quatro bytes e compatível com o ASCII e o ASCII estendido.
- O conjunto ASCII, chamado agora de Basic Latin está incluído no Unicode com os 25 bits superiores zerados.

Unicode

- Cada caractere ou símbolo no Unicode é definido por um número de 32 bits.
  - O código pode definir até $2^{32} = 4\ 294\ 967\ 296$ caracteres ou símbolos.
- A representação usa dígitos hexadecimais no seguinte formato

$$U + XXXXXXXX$$

  com cada X sendo um dígito hexadecimal.
- Como o Unicode padrão utiliza 4 bytes por caractere foi criado um código *multibyte*, que usa um número variável de bytes por caracteres.
  - Ele pode usar de 1 a 4 bytes dependendo do caractere.
- O código mais usado é o UTF-8, onde os primeiros 128 caracteres coincidem com os do ASCII e usam apenas 1 byte. Os demais caracteres usam mais bytes em sua representação.

Codificação UTF-8 – parcial

código Unicode	caractere	código UTF-8
U+0021	!	21
U+002D	-	2D
U+0030	0	30
U+0039	9	39
U+0041	A	41
U+0042	B	42
U+0061	a	61
U+0062	b	62
U+00C0	À	C380
U+00E3	ã	C3A3
U+00E7	ç	C3A7
U+00E9	é	C3A9
U+00FF	ÿ	C3BF

Caracteres Unicode em C

- O tipo `char` pode armazenar caracteres ASCII ou UTF-8.
- C11 define os tipos `char16_t` e `char32_t` para os caracteres Unicode UTF16 e UTF32, respectivamente.
- O tipo das constantes caracteres podem ser indicadas por um prefixo `u8` para um caractere UTF8, `u` para um UTF16 e `U` para um UTF32.

```
1 char     c8 = u8'A';
2 char16_t c16 = u'A';
3 char32_t c32 = U'A';
```

Inteiros

Existem cinco tipos inteiros com sinal em C11. Muitos deles podem ter sinônimos:

Tipo	Sinônimos
`signed char`	
`int`	`signed`, `signed int`
`short`	`short int`, `signed short`, `signed short int`
`long`	`long int`, `signed long`, `signed long int`
`long long`	`long long int`, `signed long long`, `signed long long int`

Para cada tipo inteiro com sinal existe um tipo sem sinal correspondente.

Tipo	Sinônimos
`unsigned char`	
`unsigned int`	`unsigned`
`unsigned short`	`unsigned short int`
`unsigned long`	`unsigned long int`
`unsigned long long`	`unsigned long long int`

Tamanho de inteiros na memória

- Um valor do tipo `char` sempre ocupa um byte.
- C somente define tamanhos de armazenamento mínimos para os demais tipos:
  - `short` ocupa no mínimo 2 bytes;
  - `long` ocupa no mínimo 4 bytes;
  - `long long` ocupa no mínimo 8 bytes;
- Embora os tipos inteiros possam ser maiores que seus tamanhos mínimos, eles são implementados tal que

  $$\texttt{sizeof(short)} \leq \texttt{sizeof(int)} \leq \texttt{sizeof(long)} \leq \texttt{sizeof(long long)}$$

- O tipo `int` deve possuir o mesmo tamanho e formato de bits que um registrador da CPU da arquitetura alvo.

Tamanhos e valores de inteiros

A tabela abaixo mostra valores típicos para os tipos inteiros. Mas, lembre-se que isto depende do hardware físico e pode mudar.

Tipo	Tamanho em bytes	Valor mínimo	Valor máximo
`char`	1	0	255
`unsigned char`	1	0	255
`signed char`	1	−128	127
`short int`	2	−32 768	32 767
`unsigned short`	2	0	65 535
`int`	4	−2 147 483 648	2 147 483 647
`unsigned int`	4	0	4 294 967 295
`long int`	4	−2 147 483 648	2 147 483 647
`unsigned long`	4	0	4 294 967 295
`long long int`	8	−9 223 372 036 854 775 808	9 223 372 036 854 775 807
`unsigned long long`	8	0	18 446 744 073 709 551 615

Tamanhos e intervalos de tipos

Para saber o tamanho exato de um tipo ou variável em C, usamos o operador `sizeof`. A expressão

```
sizeof(tipo)
```

retorna o tamanho em bytes do tipo dado como argumento e

```
sizeof expressão
```

retorna o tamanho em bytes do tipo inferido da expressão.

C possui uma biblioteca chamada *limits.h* que contém informações sobre os intervalos de inteiros. Por exemplo, o valor mínimo para um inteiro é definido lá como `INT_MIN` e o valor máximo está em `INT_MAX`. Os intervalos para outros tipos numéricos também estão na biblioteca.

Exemplo de uso de limits.h

O programa a seguir usa a *limits.h* para mostrar os valores máximos e mínimos dos tipos `char` e `int`.

```
 9 #include <stdio.h>
10 #include <limits.h> // contém as definições de CHAR_MIN, INT_MIN, etc
11
12 int main(void)
13 {
14   printf("Tamanhos em bytes e intervalos de valores dos tipos char e int\n\n
         ");
15   printf("O tipo char é%s.\n\n", CHAR_MIN < 0 ? "signed" : "unsigned");
16
17   printf(" Tipo Tamanho em bytes Mínimo       Máximo\n"
18          "-----------------------------------------------\n");
19   printf(" char %8zu %20d %15d\n", sizeof(char), CHAR_MIN, CHAR_MAX);
20   printf(" int %8zu %20d %15d\n", sizeof(int), INT_MIN, INT_MAX);
21
22   return 0;
23 } // main
```

Nota: Com **printf** o formato **s** é necessário para imprimir uma string, enquanto o formato **zu** é necessário para imprimir o tamanho em bytes como um número decimal sem sinal.

Tipos em ponto flutuante

- O C11 reconhece três tipos de ponto flutuantes: real, imaginário e complexo.
- Assim como na biblioteca limits para valores inteiros, existe uma biblioteca padrão *float.h* para valores em ponto flutuante.
- Diferentemente dos tipos integrais, os valores do tipo real são sempre com sinal.

Real

- C11 possui três tamanhos diferentes de tipos reais:
  - `float` para valores com precisão simples.
  - `double` para valores com precisão dupla.
  - `long double` para valores com precisão estendida.
- Independentemente do tamanho da máquina, C11 requer que a relação a seguir seja verdadeira

  `sizeof(float)` $\leq$ `sizeof(double)` $\leq$ `sizeof(long double)`
- Operações aritméticas envolvendo números em ponto flutuante são geralmente realizadas com precisão dupla ou maior.
- A tabela abaixo mostra os intervalos de valores e a precisão dos tipos em ponto flutuante usando notação decimal:

Tipo	Tamanho em bytes	Intervalo de valores	Menor valor positivo	Precisão em dígitos
`float`	4	± 3.4E+38	1.2E-38	6
`double`	8	± 1.7E+308	2.3E-308	15
`long double`	10	± 1.1E+4932	3.4E-4932	19

Exemplo de uso de float.h

O programa a seguir usa a *float.h* para mostrar os valores máximo e mínimo dos tipo `float`, além da precisão.

```
 9 #include <stdio.h>
10 #include <float.h>
11
12 int main(void)
13 {
14   puts("Características do tipo float\n");
15   printf("Tamanho na memória: %zu\n"
16          "Menor valor positivo: %E\n"
17          "Maior valor positivo: %E\n"
18          "Precisão: %d dígitos decimais\n",
19          sizeof(float), FLT_MIN, FLT_MAX, FLT_DIG);
20
21   puts("\nUm exemplo de precisão com float:\n");
22   double d_var = 12345.6;    // variável do tipo double
23   float f_var = (float) d_var; // inicializa a variável float com d_var.
24   printf("O número em ponto flutuante %18.10f\n", d_var);
25   printf("foi armazenado em uma variável\n"
26          "do tipo float como o valor %18.10f\n", f_var);
27   printf("O erro de arredondamento é%18.10f\n", d_var - f_var);
28
29   return 0;
30 } // main
```

Saída do exemplo anterior

```
Características do tipo float

Tamanho na memória: 4
Menor valor positivo: 1.175494E-38
Maior valor positivo: 3.402823E+38
Precisão: 6 dígitos decimais

Um exemplo de precisão com float:

O número em ponto flutuante     12345.6000000000
foi armazenado em uma variável
do tipo float como o valor      12345.5996093750
O erro de arredondamento é          0.0003906250
```

Deste exemplo note que o valor representável mais próximo do decimal 12345.6 é 12345.5996093750. Essa diferença se deve ao fato que a representação binária interna do decimal 12345.6 não é exata.

Tipo Complexo

- Um número imaginário é um número real multiplicado por $\sqrt{-1} = i$.
- O tipo complexo possui três tamanhos: `float complex`, `double complex` e `long double complex`.
- Para usar o tipo complexo deve-se incluir a biblioteca *complex.h*.

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

int main(void)
{
    double complex i = 2.0*I; // imaginário puso
    double f = 1.0;           // real puro
    double complex z = f + i; // número complexo
    printf("z = %.1f + %.1fi\n", creal(z), cimag(z));

    return 0;
}
```

Para compilar use:

```
gcc -std=c11 complexo.c -o complexo.exe -lm
```

Variáveis

- Variáveis em C são endereços de memória que possuem um tipo.
- Cada variável no programa deve ser declarada e definida.
- Em C, a *declaração* serve para dar nome a um objeto e informa o seu tipo. Enquanto que a *definição* serve para criar o objeto.
- Todas as variáveis usadas devem ser previamente declaradas.
- O tipo da variável pode ser qualquer um dos tipos permitidos em C, excetuando o tipo `void`.

Exemplos de declarações e inicializações:

```
char codigo = 'B';
int i = 14;
long long divida_interna = 1000000000000;
float juros = 14.25;
double pi = 3.1415926536;
```

Constantes

- Constantes são valores de dados que não podem ser alterados durante a execução de um programa.
- Como nas variáveis, uma constante possui um tipo.
- Discutiremos agora os seguintes tipos de constantes: booleano, caracter, inteiro, real, complexo e string.

Constante booleana

- Um tipo de dados booleano possui dois valores: `true` e `false`.
- Para usarmos estes valores devemos incluir a biblioteca *stdbool.h*.

```
#include <stdbool.h>

bool verdadeiro = true;
bool falso = false;
```

Constante caracter

- Caracteres constantes são colocados entre apóstrofes.
- Algumas vezes precisamos colocar a contrabarra (\) antes de um caracter que não possua representação gráfica associada com ele.

```
1 char c1 = 'a';
2 char c2 = '\n';
3 char c3 = '\t';
4 char c4 = '\\';
```

Constantes inteiras

Abaixo encontramos alguns exemplos de constantes inteiras.

Representação	Valor	Tipo
+123	123	`int`
-378	-378	`int`
-32271L	-32 271	`long int`
76542LU	76 542	`unsigned long int`
12789845LL	12 789 845	`long long int`

Constantes reais

Em C a forma padrão de constantes reais é *double*. Se quisermos que seja *float* ou *long double* devemos especificar na própria constante esta informação. Abaixo encontramos alguns exemplos de constantes reais

Representação	Valor	Tipo
0.	0.0	`double`
.0	0.0	`double`
2.0	2.0	`double`
3.1416	3.1416	`double`
-2.0f	-2	`float`
3.1415926536L	3.1415926536	`long double`

Constantes complexas

Em C a forma padrão de constantes complexas é *double*. Se quisermos que seja *float* ou *long double* devemos especificar na própria constante esta informação. Abaixo encontramos alguns exemplos de constantes reais

Representação	Valor	Tipo
12.3 + 14.4 * I	12.3 + 14.4i	`double complex`
14F + 16F * I	14 + 16i	`float complex`
1.4736L + 4.56756L * I	1.4736 + 4.56756i	`long double complex`

As duas componentes de uma constante complexa devem ter a mesma precisão, ou seja, se a parte real é do tipo *double* então a parte imaginária também deve ser do tipo *double*.

Constantes strings

- Uma constante string é uma sequência de zerou ou mais caracteres entre aspas.
- Abaixo estão alguns exemplos de strings.

```
""
"h"
"Oi Mundo\n"
"COMO CHOVE"
"Bom dia!"
L"Esta string contém caracteres longos"
```

Codificação de constantes

- Neste e nos próximos slides vamos discutir três formas diferentes de codificar constantes nos programas: constantes literais, constantes definidas e constantes de memória.
- Um *literal* é uma constante sem nome usada para especificar dados. Ela é o próprio valor do dado.
- No exemplo a seguir, 5 é um literal inteiro.

```
a = b + 5;
```

Constantes definidas

- Uma outra forma de escrever uma constante é usar a diretiva `#define` do pré processador do C.
- Uma diretiva *define* típica poderia ser

```
#define PI 3.1415
```

- No arquivo onde esta constante for definida, o pré processador procurará todas as ocorrências de `PI` e as substituirá por `3.1415`.
- Podemos ter mais que um uso de *define* no mesmo arquivo:

```
#define VALOR_MAXIMO 300
#define VALOR_MINIMO 100
```

Constantes de memória

- A terceira forma de usar uma constante é usar constantes de memória.
- Estas constantes usam o *qualificador de tipo* `const` para indicar que o dado não pode ser alterado.
- O formato é

  ```
  const tipo identificador = valor;
  ```

- O código a seguir cria uma constante de memória `cPi`:

  ```
  const float cPi = 3.14159;
  ```

Exemplo de uso de constantes

No programa abaixo é demonstrado as três formas de de codificar `pi` como uma constante.

```c
#include <stdio.h>

#define PI 3.1415926536

int main(void) {
    const double cPi = PI;

    printf("Constante definida PI: %f\n", PI);
    printf("Constante de memória cPI: %f\n", cPi);
    printf("Constante literal:     %f\n", 3.1415926536);
    return 0;
} // main
```

A saída deste programa é

```
Constante definida PI:    3.141593
Constante de memória cPI: 3.141593
Constante literal:        3.141593
```

Entrada e saída

- Já usamos vários vezes a função `scanf` para ler dados do teclado e a função `printf` para mostrar dados na tela.
- Aqui vamos mostrar por meio de exemplos os seus usos.

Exemplos de saídas

1. `printf("%d%c%f", 23, 'z', 4.1);`

```
23z4.100000
```

2. `printf("%d %c %f", 23, 'z', 4.1);`

```
23 z 4.100000
```

3.

```
int num1 = 23;
char ze = 'z';
floar num2 = 4.1;
printf("%d %c %f",num1, ze, num2);
```

```
23 z 4.100000
```

Exemplos de saídas

4.
```
printf("%d\t%c\t%5.1f\n", 23, 'Z', 14.2);
printf("%d\t%c\t%5.1f\n", 107, 'A', 53.6);
printf("%d\t%c\t%5.1f\n", 1754, 'F', 122.0);
printf("%d\t%c\t%5.1f\n", 3, 'P', 0.1);
```

```
23      Z       14.2
107     A       53.6
1754    F      122.0
3       P        0.1
```

5. `printf("O número%d é meu número favorito.", 23);`

```
O número23 é meu número favorito.
```

6. `printf("O número é %6d", 23);`

```
O número é    23.
```

Exemplos de saídas

7. `printf("O juros foi de %6.2f no ano passado.", 233.12);`

```
O juros foi de 233.12 no ano passado.
```

8. `printf("O juros foi de %8.2f no ano passado.", 233.12);`

```
O juros foi de  233.12 no ano passado.
```

9. `printf("O juros foi de %08.2f no ano passado.", 233.12);`

```
O juros foi de 00233.12 no ano passado.
```

10. `printf("\"%8c   %d\"", 'h', 23);`

```
"       h   23"
```

Exemplos de saídas

11

```
1 printf("Esta linha some.\r...Uma nova linha\n");
2 printf("Um caracter null\0mata o resto da linha\n");
3 printf("\nIsto é\'oi\' entre apóstrofes\n");
4 printf("Isto é\"oi\" entre aspas\n");
5 printf("\nIsto é\\ o próprio caracter de escape\n");
```

```
...Uma nova linha
Um caracter null
Isto é 'oi' entre apóstrofes
Isto é "oi" entre aspas

Isto é \ o próprio caracter de escape
```

12 `printf("|%-+8.2f|\n", 1.2);`

```
|+1.20  |
```

Para saber mais

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.
- IEEE 754. In: *Wikipedia, The Free Encyclopedia*. Disponível em `https://en.wikipedia.org/w/index.php?title=IEEE_754&oldid=840612631`.
- ASCII. In: *WIKIPÉDIA, a enciclopédia livre*. Disponível em: `https://pt.wikipedia.org/w/index.php?title=ASCII&oldid=52053391`

Fontes

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.
- IEEE 754. In: *Wikipedia, The Free Encyclopedia*. Disponível em `https://en.wikipedia.org/w/index.php?title=IEEE_754&oldid=840612631`.
- ASCII. In: *WIKIPÉDIA, a enciclopédia livre*. Disponível em: `https://pt.wikipedia.org/w/index.php?title=ASCII&oldid=52053391`