

## Introdução aos caracteres

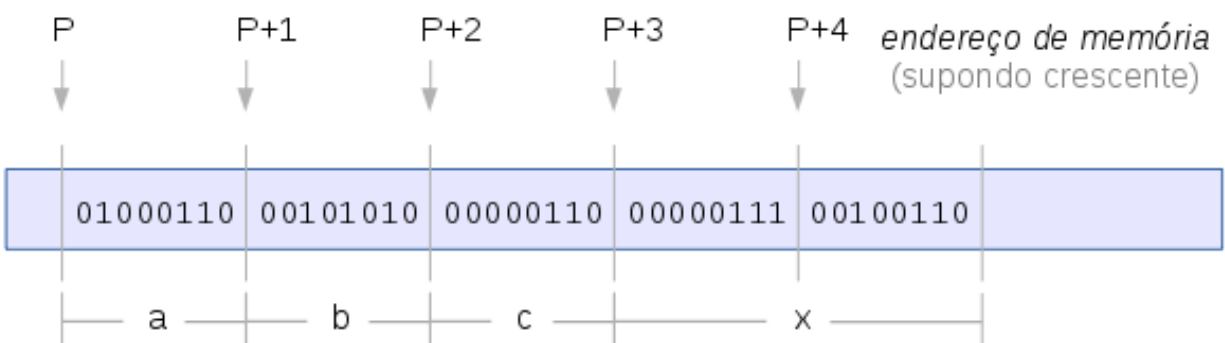
[ [Erros](#) | [Código ASCII](#) | [Entrada/saída](#) ]

Nesta seção examinaremos resumidamente como utilizar caracteres tanto em C quanto em *Python*.

### Interpretando sequência de *bits* como número ou caractere

Como citado na seção sobre [variáveis](#), a informação básica nos computadores digitais é uma sequência de *bits* e o contexto é quem indica como estes *bits* devem ser interpretados. exemplo, a sequência 1000001 poderia ser interpretada como um valor inteiro (neste caso seria o valor  $65 = 1*2^6 + 1*2^0$ ) ou como um caractere (nesse caso a letra A, vide abaixo explicação sobre código ASCII). De modo semelhante, se a sequência 110000 deve ser interpretada como caractere, seria o dígito 0, por outro lado, se for interpretada como inteiro (desconsiderando a questão do sinal), seria o  $48 = 1*2^5 + 1*2^4 = 32 + 16$ . Ou seja, existe uma tabela associando valor numérico e caractere, como explicado na seção abaixo.

Resumidamente o tratamento de dados em um computador digital, se resume à processar sequências de dígitos. Assim, ao tentar examinar determinado dado em memória, deve-se saber em que posição da memória o dado está e quantos *bits* tal dado usa. A figura ilustra isso, com 3 variáveis de tamanho 1 *bytes* (*a*, *b* e *c*) e uma ocupando 2 *bytes* (*x*). No caso de caracteres (símbolos que devem ser apresentados, como é o caso dos *dígitos* e das *letras*), basta existir uma *associação* (ou *mapeamento*) entre os *bits* e o símbolo a ser mostrado. Claro que está associação depende também do tamanho, quanto *bits* são usados para representar cada símbolo. Na seção seguinte apresento a primeira tabela de associação entre *bits* e símbolos, a [tabela ASCII](#).



*Fig. 1. Representação da memória, com 3 variáveis usando 1 bytes (8 bits) e outra usando 2 bytes.* Mas o tratamento de números *reais* apresentam questões mais difíceis, para começar não existem valores que, por maior que seja a precisão do equipamento, seria impossível representá-los. Um exemplo simples é o valor correspondente à fração  $1/3$ , que é uma *dízima periódica*, portanto com dígitos (0.3333...). Além disso existe a dificuldade de representar número grande e número muito próximos de zero. O truque usual para *simular* reais é utilizar

uma representação em *ponto flutuante* (se desejar, estude esse tópico no texto sobre [representação em ponto flutuante](#)).

## Código ASCII

Um primeiro padrão para representar os caracteres é o [American Standard Code for Information Interchange](#). O padrão ASCII é bastante antigo, ainda da década de 1960 e por isso utiliza apenas 8 *bits* para representar todos os possíveis caracteres, o que resultava em um número reduzidíssimo de 512 códigos/símbolos distintos.

Como o código ASCII foi baseado no alfabeto em língua Inglesa, eles apresentam apenas caracteres sem acentos. Usualmente associamos um natural a cada símbolo, assim no ASCII pode-se associar caracteres aos **códigos ASCII** 0 até 512.

Como exemplos de código ASCII a letra 'A' é associada ao natural 65, o 'B' ao 66 e assim por diante. Do mesmo modo, o 'a' é associada ao natural 97 e o caractere 'z' ao 122. Já o natural 48 é associado do dígito '0', o 49 ao '1' e assim por diante até o 57 associado '9'.

## Entrada e saída de caracteres em C e em Python

Como já citado nas outras seções, o contexto deve definir como sequências de *bits* devem ser tratadas. Para tratar como caractere deve-se indicar isso ao computador de alguma forma. Tanto a linguagem C como a linguagem *Python* dispõe um tipo especial de variável para tratar com caracteres.

Na linguagem C tem o tipo `char`, que precisa do formatador `%c`, mas pode-se imprimir uma variável `char` como inteiro e vice-versa, neste caso usa-se seu código ASCII. Já em *Python 2* é preciso de uma função especial para leitura (a `raw_input()`) e na impressão não é preciso qualquer diferenciação, pois *Python* aplica um filtro para tratar se a variável sendo impressa é inteira, flutuante ou caractere. Vide [tabela 1](#) abaixo.

Tab. 1. Sobre a leitura de caracteres em C e em Python

C	Python 2	Python 3
<pre>char x; // declaracao de variavel em "ponto flutuante" scanf("%c", &amp;x); // leia como "caractere" printf("%c", x); // imprima como "caractere"</pre>	<pre># Vale notar: se o usuario digitar uma "string", como # "nome", a variavel x recebera' a "string" toda x = raw_input(); # leia como "caractere" print(x);</pre>	<pre># Vale notar: se o usuario digitar uma "string", como # "nome", a variavel x recebera' a "string" toda x = input(); # leia como "caractere" print(x);</pre>

Vale notar que para atribuir constantes do tipo caractere, é necessário que um único caractere esteja cercado por aspas simples. Por exemplo, para atribuir a constante 'a' à uma variável do tipo caractere de nome `varc` deve-se usar: `varc = 'a'`; (o finalizador ';' é opcional em *Python*).

Abaixo um exemplo de código que imprime primeiro o caractere associado ao código ASCII dado e depois o contrário. Ao rodar este código será impresso os caracteres de código 48, 49, 97, 98, 65 e 66, respectivamente '0', '1', 'a', 'b', 'A' e 'B'.

Tab. 2. Sobre conversão entre o caractere ASCII e seu código inteiro

C	Python 2
<pre>#include &lt;stdio.h&gt; int main (void) {     int // Criando 6 variaveis como     (implicitamente) do tipo "int".     val1 = 48, // Cada variavel recebera um valor     inteiro que depois sera'     val2 = 49, // interpretado como caractere.     num1 = 97,     num2 = 98,     Num1 = 65,     Num2 = 66;     char // Declarando variaveis tipo     caractere     charN1 = '0', // NOTE que para constantes do     tipo     charN2 = '1', // caractere, deve-se usar aspa     simples     Char1 = 'A',     Char2 = 'B',     char1 = 'a',     char2 = 'b';     printf(" int   codigo\n"); // '\n' usado para     forcar quebra de linha na impressao     printf(" %d   %c\n", val1, val1);     printf(" %d   %c\n", val2, val2);     printf(" %d   %c\n", num1, num1);     printf(" %d   %c\n", num2, num2);     printf(" %d   %c\n", Num1, Num1);     printf(" %d   %c\n", Num2, Num2);     printf(" int   codigo\n");     printf(" %c   %d\n", charN1, charN1);     printf(" %c   %d\n", charN2, charN2);     printf(" %c   %d\n", char1, char1);     printf(" %c   %d\n", char2, char2);     printf(" %c   %d\n", Char1, Char1);     printf(" %c   %d\n", Char2, Char2);     return 0; }</pre>	<pre>def main():     val1 = 48 # Criando 6 variaveis como     (implicitamente) do tipo "int".     val2 = 49 # Cada variavel recebera um valor     inteiro que depois sera'     num1 = 97 # interpretado como caractere.     num2 = 98     Num1 = 65     Num2 = 66     print(" int   codigo")     print(" ", val1, "   ", chr(val1)) # A funcao     chr(...) devolve     print(" ", val2, "   ", chr(val2)) # o     caractere associado ao     print(" ", num1, "   ", chr(num1)) # codigo     ASCII dado     print(" ", num2, "   ", chr(num2))     print(" ", Num1, "   ", chr(Num1))     print(" ", Num2, "   ", chr(Num2))     charN1 = '0' # NOTE que para constantes do     tipo     charN2 = '1' # caractere, deve-se usar aspa     simples     Char1 = 'A'     Char2 = 'B'     char1 = 'a'     char2 = 'b'     print(" int   codigo")     print(" ", charN1, "   ", ord(charN1)) # A     funcao ord(...) devolve     print(" ", charN2, "   ", ord(charN2)) # o     codigo ASCII do caractere dado     print(" ", char1, "   ", ord(char1))     print(" ", char2, "   ", ord(char2))     print(" ", Char1, "   ", ord(Char1))     print(" ", Char2, "   ", ord(Char2)) main()</pre>

[Leônidas de Oliveira Brandão](http://line.ime.usp.br)

<http://line.ime.usp.br>

**Alterações:**

2020/08/10: nova versão, sem a parte de "float";

2019/07/31: acerto no formato da página e em várias frases;

2017/05/01: versão inicial