Programação em C

Representação inteira em C

Agostinho Brito

2021

Representação inteira em C

- Tipos inteiros são os mais comumente utilizados para armazenar dados.
- Todo tipo ocupa um espaço e existe uma padronização na forma como os dados são guardados na memória.
- A representação mais comum para os humanos é decimal (ou base numérica decimal).
- A máquina, entretanto, lida com a chamada base numérica binária.

Representação na base 2

- $(1973)_{10} = 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$.
- Cada dígito (0 a 9) exerce um papel na notação posicional de classes (unidades simples, milhares, milhões) e ordens (unidade, dezena, centena).
- Esta representação, entretanto, diz respeito à base 10. Um número expresso na **`base 2`** (ou base binária) deve ser composto usando os dígitos 0 e 1.
- Exemplo:

$$(10010110)_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (150)_{10}$$
$$(10010110)_2 = (150)_{10}$$

- Cada dígito binário é chamado de **`bit`**, um acrônimo para *binary digit*.
- O tamanho da palavra (*word size*) é a quantidade máxima de bits que um computador pode processar. Ex: 64 bits. Entretanto, C dispõe de vários tipos de dados com tamanhos menores, que ocupam menos espaço.

Convertendo para a base 2

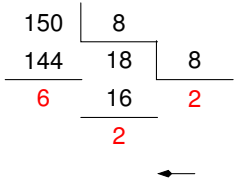
1973	10		
1970	197	10	
3	190	19	10
	7	10	1
		9	

$$(1973)_{10} = (1973)_{10}$$

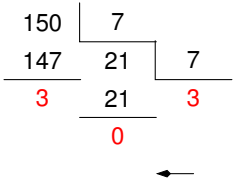
150	2						
150	75	2					
0	74	37	2				
	1	36	18	2			
		1	18	9	2		
			0	8	4	2	
				1	4	2	2
					0	2	1
						0	

$$(150)_{10} = (10010110)_2$$

Convertendo para outras bases



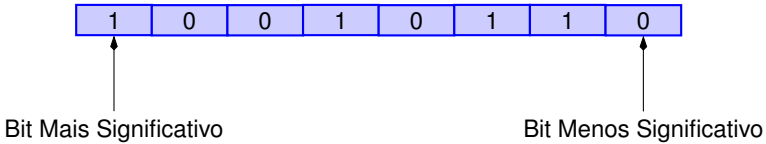
$(150)_{10} = (226)_8$

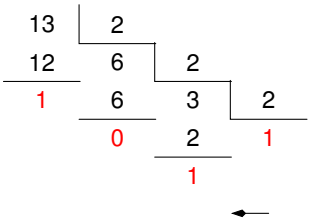


$(150)_{10} = (303)_7$

Organização dos bits em um inteiro

$$(150)_{10} = (10010110)_2$$





$$(13)_{10} = (1101)_2$$

- E quando a quantidade de bits é menor que a suportada pelo tipo?
  $(13)_{10} = (00001101)_2$ com representação `unsigned char`.
- Representação *little endian*: os bits menos significativos ficam à direita.

Números negativos

- Quando o número é negativo, o C usa uma representação denominada Complemento de 2.
- Em suma, se o número for negativo, toma-se seu módulo, inverte-se seus bits e soma-se 1 ao resultado. Ex: representar -13 em `char`.

$$(13)_{10} = (00001101)_2$$

módulo

inversão de bits

soma com 1

- $10 - 7 = 10 + (-7) = (1010)_2 + (1001)_2 = \underline{1}(0011)_2 = 3$

Convertendo entre tipos diferentes em C

- A linguagem C permite a conversão entre tipos de dados diferentes. ENTRETANTO, é preciso muito cuidado para não gerar problemas de representação.
- Considere, por exemplo, o número $(13)_{10} = (1101)_2$. Como sua representação ocupa apenas 4 bits, ele cabe dentro de qualquer tipo de dado inteiro suportado em C.
- Já o número $(269)_{10} = (100001101)_2$ precisa de pelo menos 9 bits para ser representado. Como ficaria esse número representado em uma variável do tipo `int`?

```
digite o int: 269
00000000000000000000000100001101
```

- E em uma do tipo `unsigned char`?

```
digite o char: 269
00001101
```

- Perceba que o número apresentado equivale ao decimal 13.
- Mas, $(269)_{10} = (256)_{10} + (13)_{10} = (100000000)_2 + (00001101)_2$
- Durante a leitura, apenas os bits menos significativos foram utilizados.

Convertendo entre tipos diferentes em C

- A linguagem C permite a conversão entre tipos de dados diferentes. ENTRETANTO, é preciso muito cuidado para não gerar problemas de representação.
- Considere, por exemplo, o número $(13)_{10} = (1101)_2$. Como sua representação ocupa apenas 4 bits, ele cabe dentro de qualquer tipo de dado inteiro suportado em C.
- Já o número $(269)_{10} = (100001101)_2$ precisa de pelo menos 9 bits para ser representado. Como ficaria esse número representado em uma variável do tipo `int`?

```
digite o int: 269
00000000000000000000000100001101
```

- E em uma do tipo `unsigned char`?

```
digite o char: 269
00001101
```

- Perceba que o número apresentado equivale ao decimal 13.
- Mas, $(269)_{10} = (256)_{10} + (13)_{10} = (100000000)_2 + (00001101)_2$
- Durante a leitura, apenas os bits menos significativos foram utilizados.

Convertendo entre tipos diferentes em C

- A linguagem C permite a conversão entre tipos de dados diferentes. ENTRETANTO, é preciso muito cuidado para não gerar problemas de representação.
- Considere, por exemplo, o número $(13)_{10} = (1101)_2$. Como sua representação ocupa apenas 4 bits, ele cabe dentro de qualquer tipo de dado inteiro suportado em C.
- Já o número $(269)_{10} = (100001101)_2$ precisa de pelo menos 9 bits para ser representado. Como ficaria esse número representado em uma variável do tipo `int`?

```
digite o int: 269
00000000000000000000000100001101
```

- E em uma do tipo `unsigned char`?

```
digite o char: 269
00001101
```

- Perceba que o número apresentado equivale ao decimal 13.
- Mas, $(269)_{10} = (256)_{10} + (13)_{10} = (\underline{1}00000000)_2 + (00001101)_2$
- Durante a leitura, apenas os bits menos significativos foram utilizados.

Convertendo entre tipos diferentes em C

- A conversão entre tipos é feita através do *casting* (ou coerção). É feita colocando entre parêntesis antes da expressão aritmética o tipo para o qual se deseja converter. Exemplo:

```
char c=7;
int x;
x = (int) c;
```

- Nessa expressão, antes de o valor da variável `c` ser atribuído a `x` ele é convertido para **`int`**.
- Perceba que a variável **`char`** ocupa 1 byte ao passo que **`int`** ocupa 4 bytes (Linux). Nesse caso, o valor de `c` é copiado nos bits menos significativos de `x`.
- Mas...

```
char c;
int x=269;
c = (char) x;
```

- $c \leftarrow 13$! Perceba que apenas os bits menos significativos foram de `x` foram copiados para `c`.

Convertendo entre tipos diferentes em C

CUIDADO!

```
char c;
int x=269;
c = x;
```

- Quando o especificador de *casting* não é colocado, C realiza a conversão automática para o tipo onde o dado será armazenado.
- Onde mora o perigo: a conversão entre tipos pode gerar valores capazes de pôr em risco o sistema onde o *software* opera.
- Ex: Explosão do foguete Ariane 5 em 1996. Uma conversão de um tipo de dado real de 64 bits para um inteiro de 16 bits causou um prejuízo de US$370m.
- A viagem de testes foi em velocidades baixas, de sorte que a conversão coube no inteiro de 16 bits. A segunda ultrapassou a capacidade de representação.

Obrigado