Biblioteca Serial (Porta 0) para o Arduino

Não usar classe “Serial.xxx”, pois é usada a interrupção de recepção da porta serial 0.

Senão surge o erro: multiple definition of `\_\_vector\_25'

Funções para Serial 0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Fila? |
|  |  |  |  |
| void | ser\_float | (float fx, byte prec) | Não |
| void | ser\_dec32 | (long dt) | Não |
| void | ser\_dec32u | (long dt) | Não |
| void | ser\_dec32nz | (long dt) | Não |
| void | ser\_dec32unz | (long dt) | Não |
| void | ser\_hex32 | (long dt) | Não |
|  |  |  |  |
| void | ser\_dec16 | (int dt) | Não |
| void | ser\_dec16u | (word dt) | Não |
| void | ser\_dec16nz | (int dt) | Não |
| void | ser\_dec16unz | (word dt) | Não |
| void | ser\_hex16 | (word dt) | Não |
|  |  |  |  |
| void | ser\_dec8 | (byte dt) | Não |
| void | ser\_dec8u | (byte dt) | Não |
| void | ser\_dec8nz | (byte dt) | Não |
| void | ser\_dec8unz | (byte dt) | Não |
| void | ser\_hex8 | (byte dt) | Não |
| void | ser\_crlf | (byte qtd) | Não |
| void | ser\_cr | (byte qtd) | Não |
| void | ser\_lf | (byte qtd) | Não |
| void | ser\_spc | (byte qtd) | Não |
| void | ser\_str | (byte \*msg) | Não |
| void | ser\_char | (byte dt) | Não |
|  |  |  |  |
|  |  |  |  |
| void | serb\_float | (float fx, byte prec) | Sim |
| void | serb\_dec32 | (long dt) | Sim |
| void | serb\_dec32u | (long dt) | Sim |
| void | serb\_dec32nz | (long dt) | Sim |
| void | serb\_dec32unz | (long dt) | Sim |
| void | serb\_hex32 | (long dt) | Sim |
|  |  |  |  |
| void | serb\_dec16 | (int dt) | Sim |
| void | serb\_dec16u | (word dt) | Sim |
| void | serb\_dec16nz | (int dt) | Sim |
| void | serb\_dec16unz | (word dt) | Sim |
| void | serb\_hex16 | (word dt) | Sim |
|  |  |  |  |
| void | serb\_dec8 | (byte dt) | Sim |
| void | serb\_dec8u | (byte dt) | Sim |
| void | serb\_dec8nz | (byte dt) | Sim |
| void | serb\_dec8unz | (byte dt) | Sim |
| void | serb\_hex8 | (byte dt) | Sim |
| void | serb\_crlf | (byte qtd) | Sim |
| void | serb\_cr | (byte qtd) | Sim |
| void | serb\_lf | (byte qtd) | Sim |
| void | serb\_spc | (byte qtd) | Sim |
| void | serb\_str | (byte \*msg) | Sim |
| void | serb\_char | (byte dt) | Sim |
| void | serb\_esvazia\_tx | (void) | Sim |
|  |  |  |  |
| void | ser\_tx\_fila\_config | (void) |  |
| char | ser\_tx\_poe | (char dt) |  |
| char | ser\_tx\_tira | (char \*dt) |  |
| void | ser\_config | (long br) |  |
|  |  |  |  |
| void | ser\_rx\_fila\_config | (void) |  |
| char | ser\_rx\_poe | (char dt) |  |
| char | ser\_rx\_tira | (char \*dt) |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* void **ser\_float** (float f, byte prec)

No Arduino, double e float têm a mesma precisão

Escreve no LCD o float fx com prec casas após a vírgula e apresenta o sinal.

Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])

Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.

Caso ultrapasse esses os limites imprime ###, ###

O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.

Na verdade, o máximo é 999.999.999,999967. Exemplos:

999.999.999,0 🡪 imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 🡪 imprime 876543232,000000 (por causa da precisão da representação)

* void **ser\_dec32** (long dt)

Escrever no LCD 32 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec32u** (long dt)

Escrever na fila serial32 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec32nz** (long dt)

Escrever na fila serial32 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec32unz** (long dt)

Escrever na fila serial 32 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex32** (long dt)

Escrever na fila serial 32 bits em hexadecimal.

* void **ser\_dec16** (word dt)

Escrever na fila serial 16 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec16u** (word dt)

Escrever na fila serial 16 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec16nz** (word dt)

Escrever na fila serial 16 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec16unz** (word dt)

Escrever na fila serial 16 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex16** (word dt)

Escrever na fila serial 16 bits em hexadecimal.

* void **ser\_dec8** (byte dt)

Escrever na fila serial 8 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec8u** (byte dt)

Escrever na fila serial 8 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec8nz** (byte dt)

Escrever na fila serial 8 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec8unz** (byte dt)

Escrever na fila serial 8 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex8** (byte dt)

Escrever na fila serial 8 bits em hexadecimal.

* void **ser\_crlf** (byte qtd)

Colocar na fila qtd de pares CR (0xD = ‘\r’) e LF (0xA = ‘\n’.

* void **ser\_cr** (byte qtd)

Colocar na fila qtd Carriage Return (CR = 0xD = ‘\r’).

* void **ser\_lf** (byte qtd)

Colocar na fila qtd Line Feed (LF = 0xA = ‘\n’).

* void **ser\_spc** (byte qtd)

Colocar na fila qtd brancos (0x20).

* void **ser\_str** (byte \*msg)

Colocar a string apontada por msg no buffer serial.

* void **ser\_char** (byte dt)

Colocar um char no buffer serial. Gera alerta no LCD se fila serial encher.

* void **ser\_esvazia\_tx** (void)

Esvazia toda a fila de transmissão serial. Fica preso até enviar tudo.

* void **ser\_tx\_fila\_config** (void)

Inicializar fila de transmissão serial.

* char **ser\_tx\_poe** (char cha)

Colocar um byte na fila de transmissão. Retorna TRUE se conseguiu. Retorna FALSE se fila cheia.

Se fila encher, chama a função ser\_fila\_tx\_cheia() para imprimir msg de erro na serial.

* void **ser\_fila\_tx\_cheia** (void)

Imprime mensagem de erro na porta serial para indicar que a fila serial TX está cheia.

* char **ser\_tx\_tira** (char \*cha)

Coloca no ponteiro \*char o byte retirado da fila de transmissão e retorna TRUE.

Se retornar FALSE é porque a fila está vazia.

* void **ser\_rx\_fila\_config** (void)

Inicializar fila de recepção serial.

* char **ser\_rx\_poe** (char cha)

Colocar um byte na fila de recepção. Retorna TRUE se conseguiu. Retorna FALSE se fila cheia.

Se fila encher, chama a função ser\_fila\_rx\_cheia() para imprimir msg de erro na serial.

* void **ser\_fila\_rx\_cheia** (void)

Imprime mensagem de erro na porta serial para indicar que a fila serial RX está cheia.

* char **ser\_rx\_tira** (char \*cha)

Coloca no ponteiro \*char o byte retirado da fila de recepção e retorna TRUE.

Se retornar FALSE é porque a fila está vazia.

* void **ser\_config** (long br)

Configurar porta serial com o baud rate = BR. Habilitar interrupção por transmissão e recepção.

* **ISR (USART0\_TX\_vect)**

ISR: Interrupção por dado enviado, faz ser\_tx\_ok=TRUE para indicar que transmitiu um dado.

* **ISR (USART0\_RX\_vect)**

ISR: Interrupção por dado recebido, faz ser\_rx\_ok=TRUE para indicar que recebeu um dado.

*Gabarito dos registradores de configuração da porta serial USART0*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **UCSR0A** | RXC**0** | TXC**0** | UDRE**0** | FE**0** | DOR**0** | UPE**0** | U2X**0** | MPCM**0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **UCSR0B** | RXCIE**0** | TXCIE**0** | UDRIE**0** | RXEN**0** | TXEN**0** | UCSZ**0**2 | RXB8**0** | TXB8**0** |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| **UCSR0C** | UMSEL**0**1 | UMSEL**0**0 | UPM**0**1 | UPM**0**0 | USBS**0** | UCSZ**0**1 | UCSZ**0**0 | UCPOL**0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

*Tabela 8.11. Exemplos de Baud Rate para o Arduino Mega (16 MHz)*

|  |
| --- |
| Baud Rate  (bps) |
| U2Xn = 1 | |
| UBRR | Erro |
| 9.600 | 207 | 0,2% |
| 19.200 | 103 | 0,2% |
| 38.400 | 51 | 0,2% |
| 57.600 | 34 | -0,8% |
| 115.200 | 16 | 2,1% |
| 230.400 | 8 | -3,5% |
| 250.000 | 7 | 0% |

Filas circulares para transmissão e para recepção

Fila TX

volatile char ser\_tx\_fila [SER\_TX\_FILA\_TAM]; //Espaço para a fila serial

volatile byte ser\_tx\_pin, ser\_tx\_pout; //Ponteiros para usar a fila

volatile byte ser\_tx\_ok; //Indica que terminou transmissão

volatile char ser\_rx\_fila[SER\_RX\_FILA\_TAM]; //Espaço para a fila serial de RX

volatile byte ser\_rx\_pin, ser\_rx\_pout; //Ponteiros para usar a fila

volatile byte ser\_rx\_ok; //Indica que terminou recepção