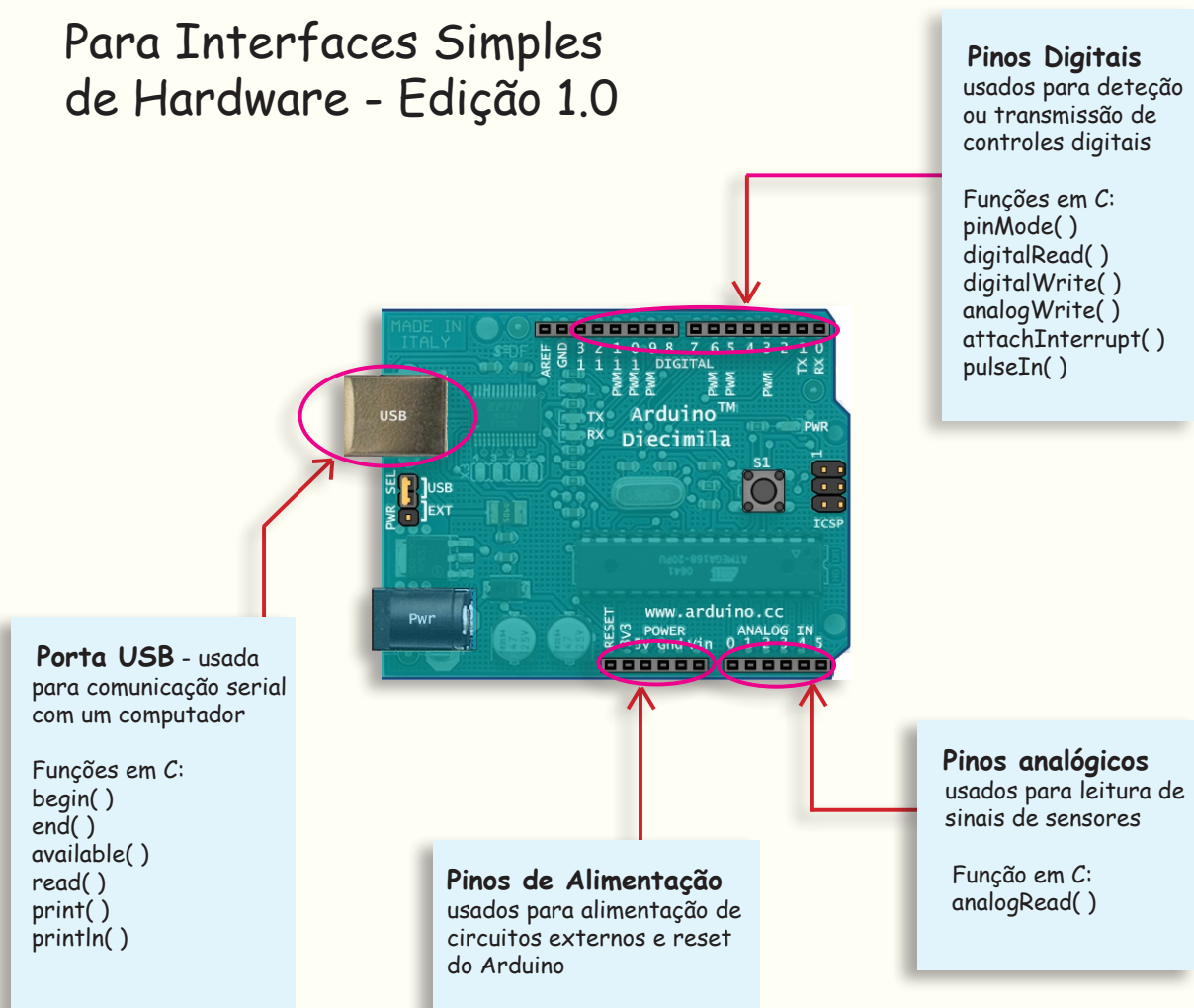


João Alexandre da Silveira

# ARDUINO

## Cartilha para Programação em C

Para Interfaces Simples  
de Hardware - Edição 1.0



Esse material é de domínio público e pode ser livremente distribuído e impresso desde que sem modificações em seu conteúdo.

[www.revistadoarduino.com.br](http://www.revistadoarduino.com.br)

Cartilha de Programação em C para o ARDUINO  
Escrita e produzida por João Alexandre da Silveira

Com informações e inspirações tomadas de:

"Arduino Programming Notebook" - Brian W. Evans  
"Experimentos com o ARDUINO" - João Alexandre da Silveira  
"Programming Interactivity" - Josua Noble  
"The C Programming Language" - Kernighan and Ritchie  
"Let Us C" - Yashavant Kanetkar  
"C for Engineers and Scientists" - Gary Bronson

[www.arduino.cc](http://www.arduino.cc)  
[www.revistadoarduino.com.br](http://www.revistadoarduino.com.br)

Edição 1.0 - janeiro de 2012

Esse material é de domínio público e pode ser livremente distribuído  
e impresso desde que sem modificações em seu conteúdo.

e-mail do autor: [planctum@yahoo.com](mailto:planctum@yahoo.com)

[www.revistadoarduino.com.br](http://www.revistadoarduino.com.br)

## PREFÁCIO

Este livreto de pouco mais de 20 páginas é um guia de consulta rápida para os iniciantes em programação do ARDUINO para criação de interfaces simples de hardware. Ele resume em tabelas, imagens e gráficos de fácil compreensão a função das portas físicas de entrada e de saída do ARDUINO e os principais comandos da sua linguagem de programação. A idéia foi reunir em um pequeno guia de bancada a descrição de cada pino do ARDUINO e as funções na sua linguagem de programação relativas a esses pinos. Também estão incluídas as funções para comunicação serial.

O texto foi dividido em duas partes: uma que trata da estrutura da linguagem de programação, e uma que trata do hardware controlado por essa programação. Nessa primeira parte é apresentada a estrutura básica da linguagem com as suas duas principais funções; a seguir uma breve introdução sobre constantes, variáveis, matrizes, funções de matemática e tempo e, por fim, as funções de controle de fluxo herdadas da linguagem C/C++. Na segunda parte as portas de entrada e saída (pinos) digitais, as portas analógicas e a porta serial do ARDUINO são descritas junto com as suas funções de controle.

Para aqueles leitores interessados em se aprofundar mais no mundo do ARDUINO e na linguagem C sugerimos os livros e sites web que aparecem na página II deste livreto. Sugestões, críticas e comentários sobre esse trabalho podem ser enviadas diretamente para o autor pelo site [www.revistadoarduino.com.br](http://www.revistadoarduino.com.br); onde o leitor poderá também cadastrar seu e-mail para ser notificado sobre uma nova edição desta Cartilha.

João Alexandre da Silveira  
janeiro de 2012

## ÍNDICE

<b>MAPA DAS ENTRADAS E SAÍDAS DO ARDUINO</b>	<b>2</b>
<b>A - A ESTRUTURA DA LINGUAGEM DO ARDUINO</b>	<b>3</b>
A Estrutura Básica	
Funções	
Declaração de funções	
setup( )	
loop( )	
Os símbolos	
{ }	
;	
//	
/*...*/	
<b>A.1 - CONSTANTES E VARIÁVEIS</b>	<b>4</b>
TRUE/FALSE	
HIGH/LOW	
INPUT/OUPUT	
Escopo da Variável	
Declaração da Variável	
Tipos de variáveis	
<b>A.2 - MATRIZES</b>	<b>5</b>
Declaração de uma Matriz	
Escrever/Ler uma Matriz	
<b>A.3 - LÓGICA E ARITMÉTICA</b>	<b>6</b>
Símbolos compostos	
Operadores de comparação	
Operadores lógicos	
<b>A.4 - FUNÇÕES MATEMÁTICAS E DE TEMPO</b>	<b>7</b>
delay( )	
delayMicroseconds( )	
millis( )	
random( )	
abs( )	
map( )	
<b>A.5 - FUNÇÕES PARA CONTROLE DE FLUXO</b>	<b>8</b>
if	
if...else	
if...else...if	

while  
do...while  
for  
switch...case  
operador ternário '?'

<b>B- AS PORTAS DE E/S DO ARDUINO E SUAS FUNÇÕES EM C</b>	<b>13</b>
B.1 - OS PINOS DIGITAIS	13
pinMode( )	
digitalRead( )	
digitalWrite( )	
analogWrite( )	
attachInterrupt( )	
pulseIn( )	
B.2 - OS PINOS ANALÓGICOS	15
analogRead( )	
B.3 - A PORTA SERIAL DO ARDUINO	16
Serial.begin( )	
Serial.end( )	
Serial.available( )	
Serial.read( )	
Serial.print( )	
Serial.println( )	
B.4 - OS PINOS DE ALIMENTAÇÃO	17

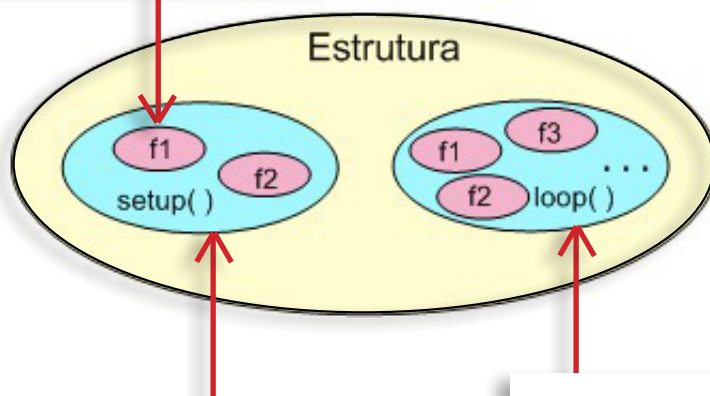


## A - A ESTRUTURA DA LINGUAGEM DO ARDUINO

A **estrutura** básica da linguagem de programação do Arduino é bastante simples; ela é formada por dois blocos de funções que carregam outros blocos de funções escritas em linguagem C/C++. O primeiro bloco de funções forma a função **setup( )**; o segundo, a função **loop( )**.

**Funções** em linguagens de programação são como sub-rotinas ou procedimentos; são pequenos blocos de programas usados para montar o programa principal. Elas são escritas pelo programador para realizar tarefas repetitivas, ou podem ser importadas prontas para o programa em forma de bibliotecas.

**Declaração da Função** toda função deve ser declarada antes de ser chamada atribuindo-lhe um tipo e um nome seguido de parênteses, onde serão colocados os parâmetros de passagem da função. Depois do nome são definidos entre as chaves '{' e '}' os procedimentos que a função vai executar.



**setup( )**: Essa é a primeira função a ser chamada quando o programa inicia. E é executada apenas nessa primeira vez. Esta é uma função de preparação: ela dita o comportamento dos pinos do Arduino e inicializa a porta serial.

**loop( )**: A função **loop( )** é chamada logo a seguir e todas as funções embarcadas nela são repetidamente executadas. Ela fica lendo os pinos de entrada do Arduino e comandando os pinos de saída e a porta serial.

Os **Símbolos** usados na construção de funções são os seguintes:

{ } - Dentro das **chaves** vão os procedimentos (statements) que a função deve executar;

;- O **ponto-e-vírgula** é usado para marcar o final de um procedimento;

// - **comentário de uma linha**: qualquer caracter depois das duas barras é ignorado pelo programa;

/\*...\*/ - **comentário em várias linhas**: qualquer texto colocado entre esses simbolos também é ignorado pelo programa.

**Exemplo:**

```

/*
Nesse código a função setup( ) ativa a porta serial em 9600 bits/s e a função loop( ) fica transmitindo a frase 'Hello World!' pela porta serial a cada 2 segundos.
*/

void setup( ) {
    Serial.begin(9600);           // inicializa a porta serial
}
void loop( ) {
    Serial.println(" Hello World! "); // transmite frase
    delay(2000);
}

```

**A.1 - CONSTANTES E VARIÁVEIS**

<p><b>CONSTANTES</b> são valores predefinidos que nunca podem ser alterados. Na linguagem C do Arduino são 3 os grupos de constantes; os dois componentes de cada grupo sempre podem ser representados pelos números binários 1 e 0.</p>	<p><b>TRUE/FALSE</b> são constantes booleanas que definem estados lógicos. Verdadeiro é qualquer valor que não seja zero. Falso é sempre o valor zero.</p>
	<p><b>HIGH/LOW</b> essas constantes definem as tensões nos pinos digitais do Arduino. Alto é uma tensão de 5 volts; baixo o terra (ou 0 volt).</p>
	<p><b>INPUT/OUTPUT</b> são constantes programadas pela função <b>pinMode( )</b> para os pinos do Arduino; eles podem ser entradas (de sensores) ou podem ser saídas (de controle).</p>
<p><b>VARIÁVEIS</b> são posições na memória de programa do Arduino marcadas com um nome e o tipo de informação que irão guardar. Essas posições podem estar vazias ou podem receber um valor inicial. Os valores das variáveis podem ser alterados pelo programa.</p>	<p><b>Escopo da Variável</b> é o limite ou abrangência da variável. Uma variável pode ser declarada em qualquer parte do programa. Se for declarada logo no início, antes da função <b>setup( )</b>, ela tem o escopo de <b>Variável Global</b>, e por isso ela pode ser vista e usada por qualquer função no programa. Se declarada dentro de uma função ela tem o escopo de <b>Variável Local</b>, e só pode ser usada por essa função.</p>
	<p><b>Declaração da Variável</b> como as funções, toda variável deve ser declarada antes de ser chamada. Essa declaração consiste em atribuir previamente um tipo e um nome à variável.</p>
	<p><b>Tipos de variáveis:</b></p> <p><b>byte</b> - esse tipo armazena 8 bits (0-255);  <b>int</b> - armazena números inteiros de até 16 bits;  <b>long</b> - armazena números inteiros de até 32 bits;  <b>float</b> - variáveis deste tipo podem armazenar números fracionários de até 32 bits.</p>



**Exemplo:**

/\* Esse programa escrito em C do Arduino aumenta e diminui gradativamente o brilho de um LED conectado no pino PWM 10 do Arduino. \*/

```
int i=0; // declaração da variável global inteira i iniciada com 0
void ledOn(); // declaração da função criada ledOn do tipo void
void setup() {
    pinMode(10,OUTPUT); // aqui 2 parâmetros são passados à função pinMode()
}
void loop() {
    for (i=0; i <= 255; i++) ledOn(); // aumenta o brilho do led
    for (i=255; i >= 0; i--) ledOn(); // diminui o brilho do led
}
void ledOn() { // função que acende o led
    analogWrite (10, i); // o nº do pino e o valor de i são passados à função analogWrite()
    delay (10);
}
```

**A.2 - MATRIZES**

<p><b>MATRIZES</b> são coleções de variáveis do mesmo tipo, portanto são posições na memória de programa, com endereços que podem ser acessados por meio de um identificador, chamado de <b>índice</b>. A primeira posição de uma matriz é sempre a de índice 0.</p>	<p><b>Declaração de uma Matriz I</b> As matrizes, como as variáveis e as funções, devem ser declaradas com um tipo e um nome seguido de colchetes; e podem também ser inicializadas com os valores entre as chaves. Exemplo:</p> <pre>int nomeMatriz [ ] = { 16,32,64,128, ... };</pre>
	<p><b>Declaração de uma Matriz II</b> Pode-se também declarar somente o tipo, o nome e o tamanho da matriz, deixando para o programa o armazenamento de variáveis nas posições, ou índices, da matriz.</p> <pre>int nomeMatriz [ 10 ] ; //nomeMatriz com dez 10 //posições para variáveis inteiras</pre>
	<p><b>Escrever/Ler uma Matriz</b> Para guardar o inteiro 16 na 4ª posição da matriz <b>nomeMatriz</b>, usa-se:</p> <pre>nomeMatriz [3] = 16;</pre> <p>Para atribuir o valor armazenado na 5ª posição de <b>nomeMatriz</b> à variável <b>x</b>:</p> <pre>int x = nomeMatriz[4];</pre>

## A.3 - LÓGICA E ARITMÉTICA

<p><b>Operações Aritméticas e lógicas</b> as 4 operações aritméticas, divisão, multiplicação, adição e subtração, são representadas pelos símbolos: /, *, + e -, respectivamente, separando os operandos. E são 3 os operadores lógicos na linguagem do Arduino que são usados para comparar duas expressões e retornar a constante TRUE/FALSE.</p>	<p><b>Símbolos compostos</b> são aqueles que combinam os símbolos aritméticos entre si e com o sinal de atribuição:</p> <pre>x ++      // x=x+1 x --      // x=x-1 x += y    // x=x+y x -= y    // x=x-y x *= y    // x=x*y x /= y    // x=x/y</pre>
	<p><b>Operadores de comparação</b> comparam uma variável com uma constante, ou variáveis entre si. São usados para testar se uma condição é verdadeira.</p> <pre>x == y    // x é igual a y x != y    // x não é igual a y x &lt; y     // x é menor que y x &gt; y     // x é maior que y x &lt;= y    // x é menor ou igual a y x &gt;= y    // x é maior ou igual a y</pre>
	<p><b>Operadores lógicos</b> são usados para comparar duas expressões, retornam 1 ou 0 (verdadeiro/falso).</p> <pre>&amp;&amp;      AND   porta lógica 'E'         OR    porta lógica 'OU' !       NOT   porta lógica NÃO</pre>

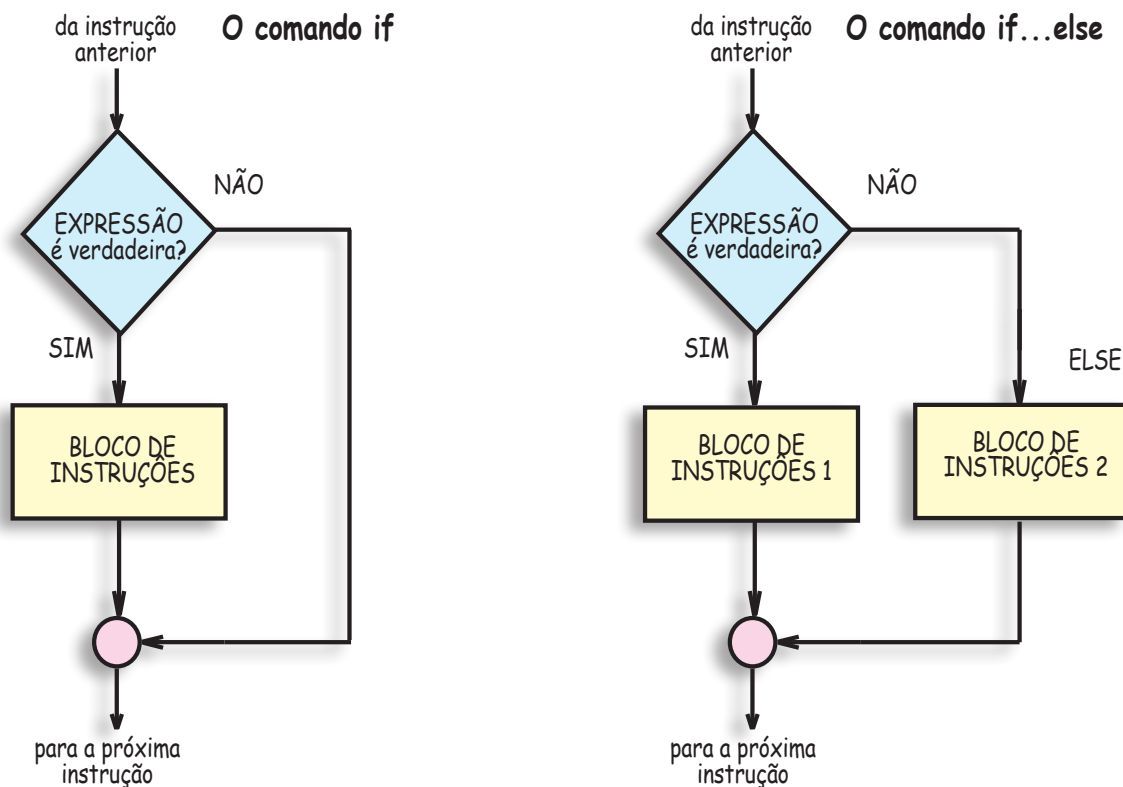
## A.4 - FUNÇÕES MATEMÁTICAS E DE TEMPO

Função	Exemplo	Notas
<b>delay(ms)</b> Essa função pausa o programa por um período em milissegundos indicado pelo parâmetro entre parênteses.	<b>delay(1000);</b> Com esse parâmetro o programa vai pausar durante 1 segundo (1000 ms).	Durante o período em que essa função está ativa qualquer outra função no programa é suspensa; é equivalente ao HALT em Assembly. Somente as interrupções de hardware podem parar essa função.
<b>delayMicroseconds(us)</b> Essa função pausa o programa por um período em microssegundos indicado pelo parâmetro entre parênteses.	<b>delayMicroseconds(1000);</b> Com esse parâmetro o programa vai pausar durante 1 ms (1000 us).	As mesmas observações acima para a função <b>delay(ms)</b> são válidas aqui.
<b>millis( )</b> Retorna o número de milissegundos desde que o Arduino começou a executar o programa corrente.	<b>long total = millis( );</b> Aqui a variável inteira longa (de 32 bits) 'total' vai guardar o tempo em ms desde que o Arduino foi inicializado.	Essa variável vai ser resetada depois de aproximadamente 9 horas.
<b>random(min, max)</b> Gera números pseudo-aleatórios entre os limites <b>min</b> e <b>max</b> especificados como parâmetros.	<b>int valor = random(100, 400);</b> A variável 'valor' vai ser atribuído um número inteiro qualquer entre 100 e 400.	O parâmetro <b>min</b> é opcional e se excluído o limite mínimo é 0. No exemplo variável 'valor' poderá ser qualquer número inteiro entre 0 e 400.
<b>abs(x)</b> Retorna o módulo ou valor absoluto do número real passado como parâmetro.	<b>float valor = abs(-3.14);</b> À variável 'valor' vai ser atribuído o número em ponto flutuante (e sem sinal) 3.14.	
<b>map(valor, min1, max1, min2, max2)</b> A função <b>map( )</b> converte uma faixa de valores para outra faixa. O primeiro parâmetro 'valor' é a variável que será convertida; o segundo e o terceiro parâmetros são os valores mínimo e máximo dessa variável; o quarto e o quinto são os novos valores mínimo e máximo da variável 'valor'.	<b>int valor = map(analogRead(A0), 0, 1023, 0, 255);</b> A variável 'valor' vai guardar a leitura do nível analógico no pino A0 convertida da faixa de 0-1023 para a faixa 0-255.	Com essa função é possível reverter uma faixa de valores, exemplo: <b>int valor = map(x, 1, 100, 100, 1);</b>

## A.5 - FUNÇÕES PARA CONTROLE DE FLUXO

**if** é um controle de fluxo usado para selecionar uma ou mais instruções baseado no resultado de um teste de comparação. Todas as instruções entre as chaves { e } são executadas somente se o resultado desse teste for verdadeiro; se não, essas instruções não são executadas. Verdadeiro é qualquer resultado, mesmo negativo, diferente de zero. Falso é um resultado zero.

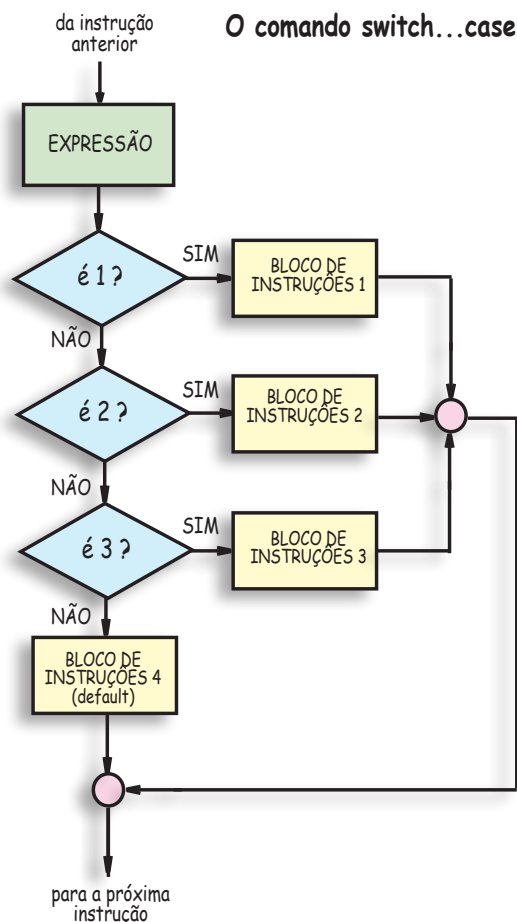
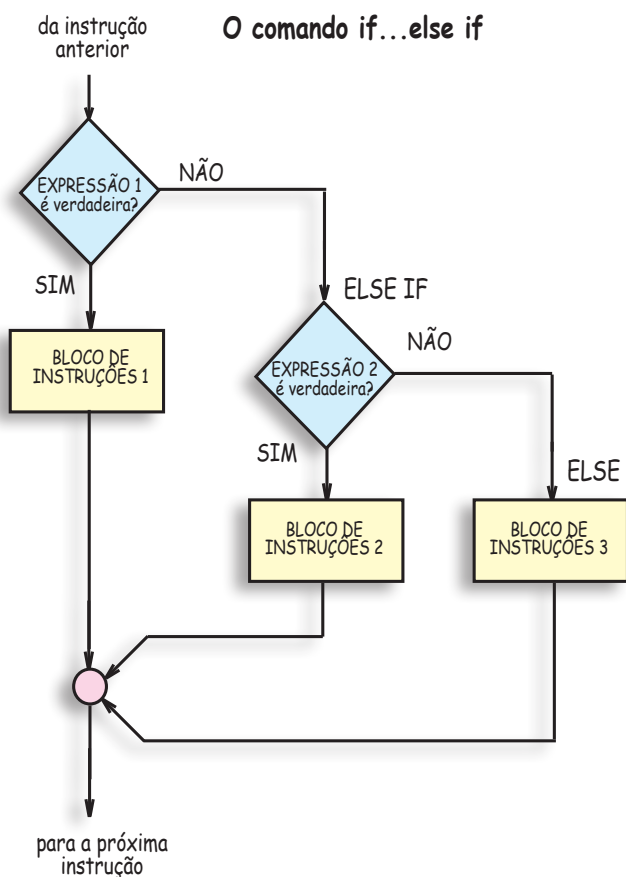
```
if (expressão) {
    bloco de instruções; // se 'expressão' for verdadeira, 'bloco de instruções' é executado
}
```



**if...else** Ao se acrescentar mais um **bloco de instruções** no loop do comando if pode-se criar o comando **if...else**, para fazer um teste novo quando o resultado da expressão for falsa.

```
if (expressão) {
    bloco de instruções1; // se 'expressão' for verdadeira, 'bloco de instruções1' é executado
}
else {
    bloco de instruções2; // se 'expressão' for falsa, 'bloco de instruções2' é executado
}
```

**if...else if** E de novo ao se acrescentar agora o comando **if...else** no loop do comando **if** pode-se criar mais um outro comando, o **if...else if**. No exemplo abaixo se '**expressão1**' for verdadeira o '**bloco de instruções1**' é executado; se '**expressão1**' for falsa mas '**expressão2**' for verdadeira '**bloco de instruções2**' é executado; e se '**expressão1**' e '**expressão2**' forem falsas o '**bloco de instruções3**' é executado.



```

if (expressão1) {
    bloco de comandos1;
}
else if (expressão2) {
    bloco de instruções2;
}
else {
    bloco de comandos3;
}
    
```

```

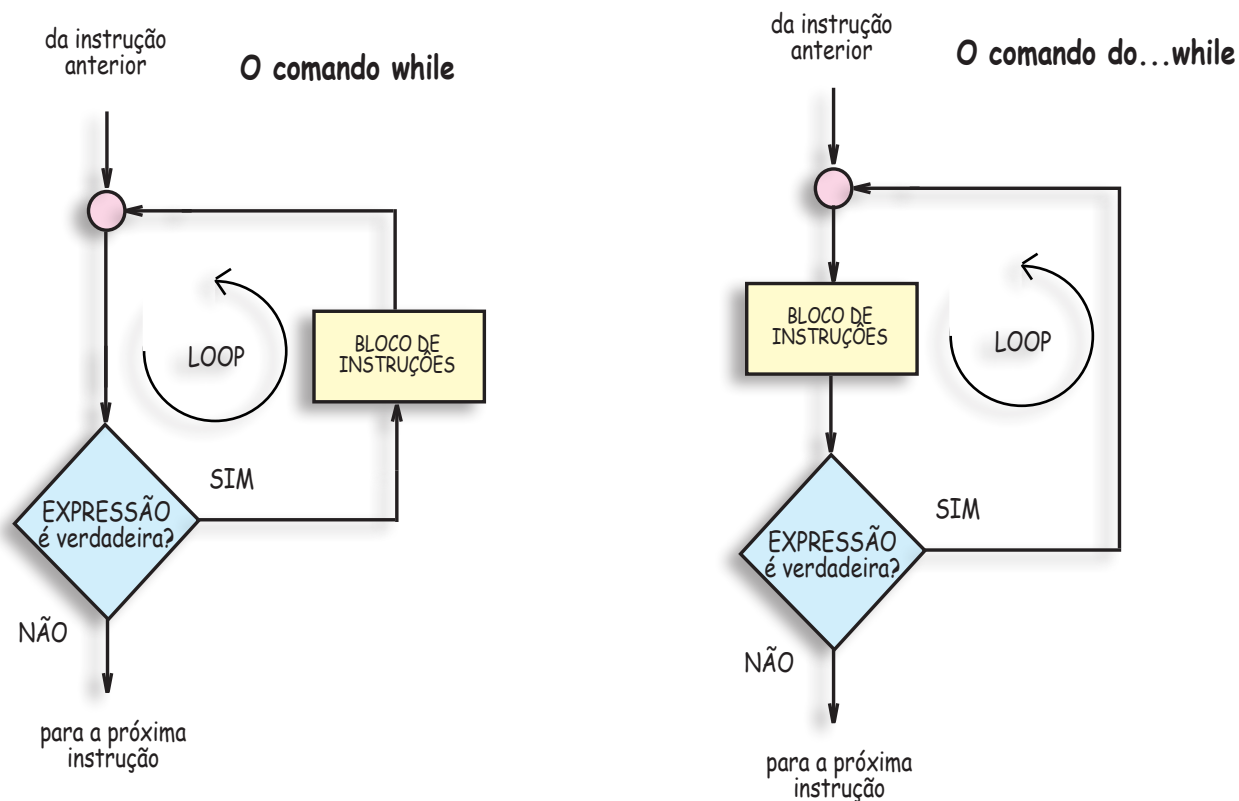
switch (expressão) {
    case 1: bloco de instruções1;
    break;
    case 2: bloco de instruções2;
    break;
    case 3: bloco de instruções3;
    break;
    default: bloco de instruções4;
}
    
```

**switch...case** É possível ir inserindo comandos **if...else** na posição do segundo bloco de instruções de outro comando **if...else** e assim criar uma cadeia de comandos para testar dezenas de expressões até encontrar uma que retorne um resultado verdadeiro e executar um dos blocos de instruções; mas existe um comando próprio que simplifica bastante essa seleção, é o comando **switch...case**. Esse comando permite comparar uma mesma variável inteira, ou uma expressão que retorne um inteiro, com vários valores possíveis.

**while** Uma das operações mais frequentes que os programas executam é repetir um grupo de instruções até que uma condição inicialmente verdadeira se torne falsa. E para isso que serve o comando **while**. A sua sintaxe é a seguinte:

```
while (expressão) {
    bloco de instruções;
}
```

O bloco de instruções será executado enquanto o parâmetro **expressão** for verdadeiro.



**do...while** Para que o bloco de instruções seja executado ao menos uma vez, ele é deslocado para a entrada da caixa de decisões, antes do teste de validade:

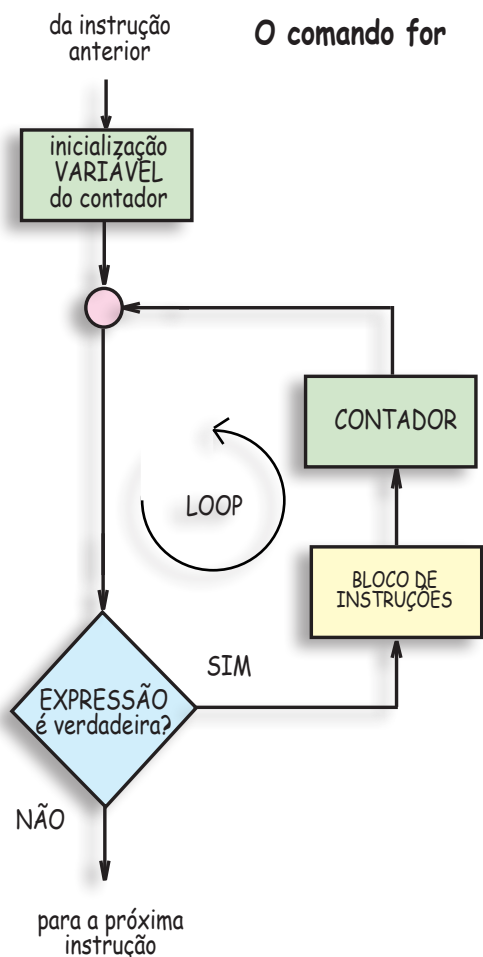
```
do {
    bloco de instruções;
} while (expressão);
```

Aqui o 'bloco de instruções' será executado primeiro e só então o parâmetro 'expressão' é avaliado.

**for** Inserindo-se no loop do comando **while** um contador que registre cada execução do **bloco de instruções** cria-se o comando **for**. Esse contador deve ter uma variável de controle que deve ser previamente inicializada com um tipo e um valor. A sua sintaxe é a seguinte:

```
for (variável; expressão; incremento) {
    bloco de instruções;
}
```

A **variável** é inicializada normalmente com 0 ou 1; o parâmetro **expressão** deve conter o valor máximo (ou mínimo) que o contador deve alcançar; e **incremento** é o valor que será incrementado (ou decrementado) da variável cada vez que o bloco de instruções é executado. Observe que cada parâmetro entre parênteses é separado por ponto e vírgula.



Exemplo:

```
for (int i = 0; i <= 10; i++) {
    println (contador);
    delay(1000);
}
```

Nesse exemplo a variável de controle *i* do contador é inicializada com 0; o contador é testado e se o valor nele acumulado for menor que 10 seu valor é enviado para o Terminal, e depois de 1 segundo, o contador é incrementado e seu novo valor é testado novamente.

**O operador ternário '?'** É possível simplificar códigos com comandos `if...else` em C/C++ com o operador condicional '?', também chamado de operador ternário. Esse operador avalia uma expressão e se esta for verdadeira uma instrução é executada, se a expressão for falsa uma outra expressão é executada. A sua sintaxe é a seguinte:

**(expressão) ? instrução1 : instrução2;**

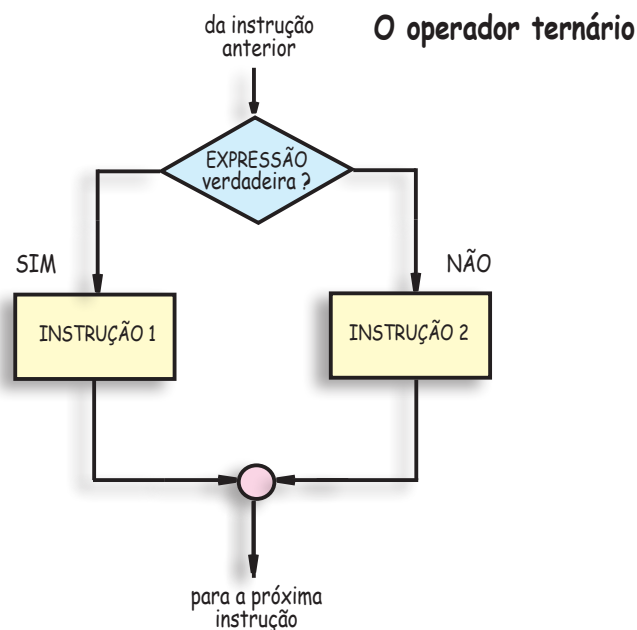
Note o uso e a posição entre as duas instruções de dois pontos na sintaxe desse operador.

Exemplo:

```
int x = 8;  
y = (x > 10) ? 15 : 20;
```

Aqui o valor de `y` vai depender da avaliação da expressão do operador ternário; como o valor de `x` vale 8, a expressão `(x>10)` é falsa, por isso o inteiro 20 será atribuído a `y`; se o valor atribuído a `x` fosse maior que 10, `y` seria 15. Essa mesma expressão com o comando `if...else` ficaria assim:

```
int x = 8;  
if (x > 10) {  
  y = 15;  
}  
else  
  y = 20;
```



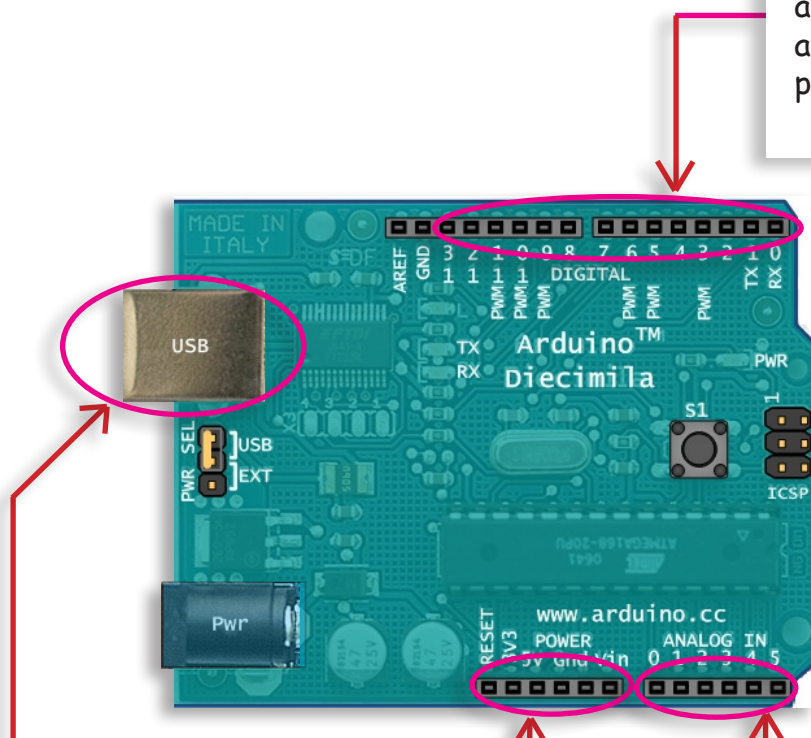


# MAPA DAS ENTRADAS E SAÍDAS DO ARDUINO

[E suas Funções na linguagem do Arduino]

**Pinos Digitais**  
usados para detecção ou transmissão de controles digitais

Funções em C:  
pinMode( )  
digitalRead( )  
digitalWrite( )  
analogWrite( )  
attachInterrupt( )  
pulseIn( )



**Porta USB** - usada para comunicação serial com um computador

Funções em C:  
begin( )  
end( )  
available( )  
read( )  
print( )  
println( )

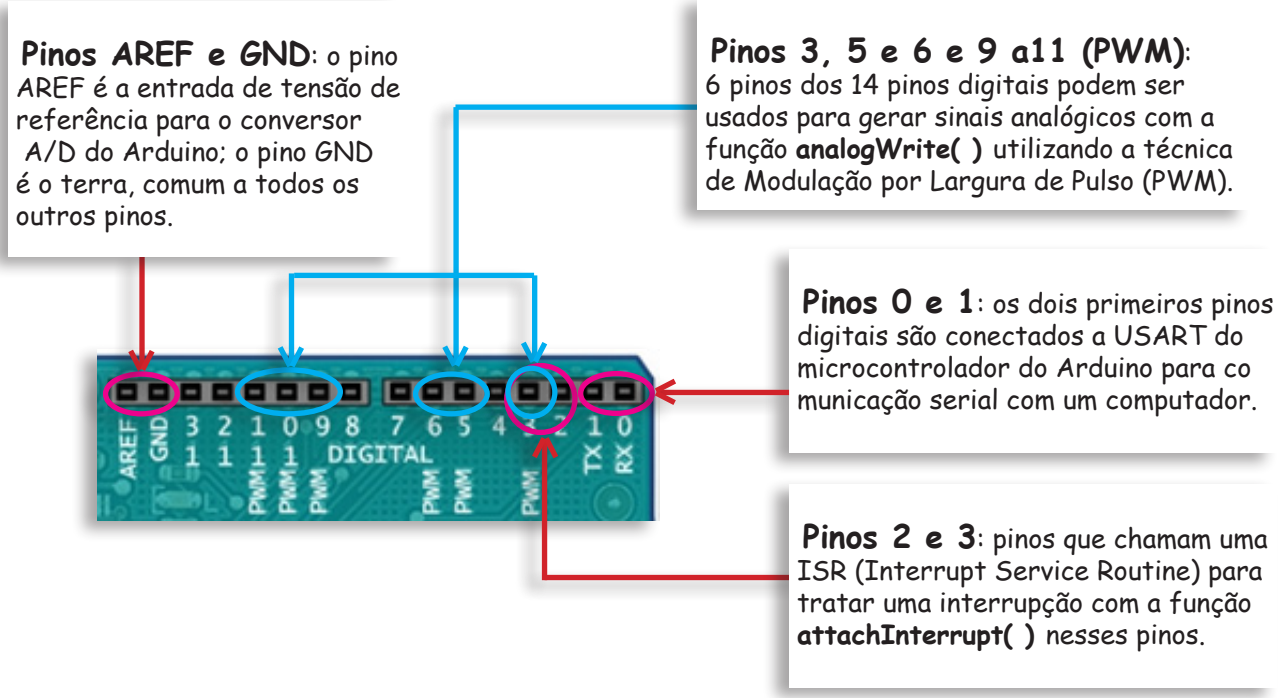
**Pinos de Alimentação**  
usados para alimentação de circuitos externos e reset do Arduino

**Pinos analógicos**  
usados para leitura de sinais de sensores

Função em C:  
analogRead( )

## B - AS PORTAS DE E/S DO ARDUINO E SUAS FUNÇÕES

**B.1 - OS PINOS DIGITAIS** São 14 pinos marcados com o nome **DIGITAL** logo abaixo de duas barras de 8 pinos. São numerados de 0 a 13 da direita para a esquerda e podem ser configurados pela função `pinMode( )` para detetarem ou transmitirem níveis lógicos digitais (verdadeiro/falso, 1/0 ou HIGH/LOW).



Função	Exemplo	Notas
<p><b>pinMode(pino, modo)</b>                      Serve para estabelecer a direção do fluxo de informações em qualquer dos 14 pinos digitais. Dois parâmetros devem ser passados à função: o primeiro indica qual pino vai ser usado; o segundo, se esse pino vai ser entrada ou se vai ser saída dessas informações.</p>	<p><b>pinMode(2, OUTPUT);</b>                      Aqui o pino 2 é selecionado para transmitir informações do Arduino para um circuito externo qualquer. Para configurar esse pino como entrada, o segundo parâmetro dessa função deve ser INPUT.</p>	<p>Essa função é sempre escrita dentro da função <code>setup( )</code>.</p>
<p><b>digitalRead(pino)</b>                      Uma vez configurado um certo pino como entrada com a função <code>pinMode( )</code>, a informação presente nesse pino pode ser lida com a função <code>digitalRead( )</code> e armazenada numa variável qualquer.</p>	<p><b>int chave = digitalRead(3);</b>                      Nesse exemplo a variável inteira 'chave' vai guardar o estado lógico (verdadeiro/falso) presente no pino digital 3.</p>	
<p><b>digitalWrite(pino, valor)</b>                      Para enviar um nível lógico para qualquer pino digital do Arduino utiliza-se essa função. Dois parâmetros são requeridos: o número do pino e o estado lógico (HIGH/LOW) em que esse pino deve permanecer.</p>	<p><b>digitalWrite(2, HIGH);</b>                      Aqui uma tensão de 5 volts é colocada no pino 2. Para enviar terra para esse pino o segundo parâmetro deverá ser LOW.</p>	<p>É necessário configurar previamente o pino como <b>saída</b> com a função <code>pinMode( )</code>.</p>

No programa abaixo essas tres funções são utilizadas para acender um led no pino 2 toda vez que uma chave normalmente aberta no pino 3 for pressionada.

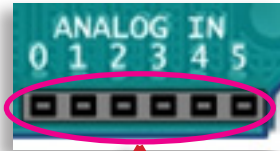
```
void setup() {
  pinMode(2,OUTPUT);      // led no pino 2 como saída
  pinMode(3,INPUT);      // chave no pino 3 como entrada
}
void loop() {
  int chave = digitalRead(3); // variável 'chave' guarda estado do pino 3
  digitalWrite(2,chave);    // estado de 'chave' é passado para pino 2.
}
```

O código dentro da função loop( ) acima pode ainda ser simplificado da seguinte forma:

```
void loop() {
  digitalWrite(2,digitalRead(3)); // led (pino 2) acende se chave (pino 3) for pressionada.
}
```

Função	Exemplo	Notas
<p><b>analogWrite(pino, valor)</b> O Arduino pode gerar tensões analógicas em 6 de seus 14 pinos digitais com a função <b>analogWrite( )</b>. Dois parâmetros devem ser passados à função: o primeiro indica em qual pino será gerada a tensão; o segundo determina a amplitude dessa tensão, e deve ter valores entre 0 (para 0 volt) e 255 (para 5 volts).</p>	<p><b>analogWrite(10, 128);</b> Com esses parâmetros uma tensão analógica de 2,5 volts vai aparecer no pino 10. Não é necessário configurar um pino PWM como saída com a função <b>pinMode( )</b> quando se chama função <b>analogWrite( )</b>.</p>	<p>Modulação por Largura de Pulsos, ou PWM (Pulse Width Modulation) na lingua inglesa, é uma técnica usada para gerar tensões analógicas a partir de uma sequência de pulsos digitais.</p>
<p><b>attachInterrupt(pino, função, modo)</b> Essa função é uma rotina de serviço de interrupção, ou ISR (Interrupt Service Routine) em inglês. Toda vez que ocorrer uma interrupção por hardware no pino digital 2 ou no 3 do Arduino uma outra função, criada pelo programador, vai ser chamada. O terceiro parâmetro, <b>modo</b>, informa como a interrupção vai ser disparada, se na borda de subida do pulso detetado no pino do Arduino, se na borda de descida, se quando o pulso for baixo ou se na mudança de nível desse pulso.</p>	<p><b>attachInterrupt(0, contador, RISING);</b> Nesse exemplo a função 'contador' vai ser chamada quando o Arduino detetar uma mudança do nível LOW para o nível HIGH em seu pino 2. Nessa ISR o parâmetro 0 monitora o pino 2, o parâmetro 1 monitora o pino 3.</p>	<p>LOW - dispara a interrupção quando o pino está em 0; CHANGE - dispara sempre que o pino muda de estado (de 0 para 1, ou vice-versa); RISING - somente quando o pino muda de 0 para 1; FALLING - somente quando o pino muda de 1 para 0.</p>
<p><b>pulseIn(pino, valor, espera)</b> Essa função mede a largura em microssegundos de um pulso em qualquer pino digital. O parâmetro 'valor' diz à função que tipo de pulso deve ser medido, se HIGH ou LOW. O parâmetro 'espera' (time out) é opcional e se passado à função faz com que a medida do pulso só comece após o tempo em microssegundos ali especificado.</p>	<p><b>pulseIn(4, HIGH);</b> Aqui essa função vai monitorar o pino 4, e quando o nível nesse pino mudar de LOW para HIGH a sua largura vai ser medida até que seu nível volte para LOW. Se, por exemplo, for passado o valor 100 como terceiro parâmetro, a medida da largura do pulso só será disparada após 100 uS.</p>	<p>Uma aplicação interessante para essas duas últimas funções pode ser vista no meu livro "Experimentos com o Arduino", no capítulo que mostra como montar um contador de dois dígitos com mostradores de 7-segmentos.</p>

**B.2 - OS PINOS ANALÓGICOS** São 6 pinos em uma só barra com o nome **ANALOG IN**, localizada no lado oposto às barras dos pinos digitais. São numerados de 0 a 5, agora da esquerda para a direita. Esses pinos são usados para leitura de sinais analógicos de sensores conectados ao Arduino, e podem ser de quaisquer valores entre zero a 5 volts. Os pinos de entradas analógicas não precisam ser previamente configurados com a função **pinMode( )**.



**Pinos 0 a 5:** esses 6 pinos aceitam tensões entre zero e 5 volts CC que vão ao conversor A/D de 10 bits no microcontrolador do Arduino. O pino AREF, numa das barras de pinos digitais, é a entrada de tensão de referência para esse conversor.

Função	Exemplo	Notas
<p><b>analogRead(pino)</b> Essa função lê o nível analógico presente no pino indicado pelo parâmetro entre parênteses e, após a conversão para o seu equivalente em bits, o guarda em uma variável determinada pelo programador.</p>	<p><b>int sensor = analogRead(A0);</b> Aqui a variável inteira 'sensor' vai armazenar a tensão analógica convertida para digital presente no pino A0. Essa informação vai ser um valor inteiro entre 0 (para 0 volt no pino) e 1023 (se 5 volts no pino). Uma tensão de 2,5 volts no pino A0 vai fazer a variável 'sensor' guardar o valor inteiro 512.</p>	<p>Os pinos analógicos são reconhecidos pela linguagem C do Arduino tanto como A0 a A5 como 14 a 19. Assim, a mesma expressão acima pode ser escrita também da seguinte forma: <b>int sensor = analogRead(14);</b></p>

Uma observação importante em relação a esses pinos analógicos é que eles podem ser configurados também como pinos digitais pela função **pinMode( )**, aumentando assim o número desses pinos para 20. Assim, a expressão **pinMode(14, OUTPUT);** transforma o pino analógico A0 em pino de saída digital como qualquer outro presente nas duas barras de pinos digitais.

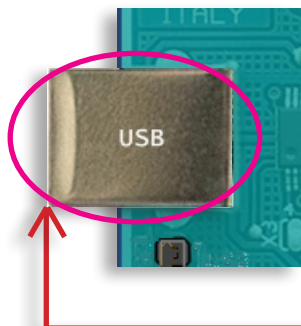
Exemplo:

O código a seguir lê uma tensão CC no pino analógico A5 (pino 19) e envia esse valor digitalizado para controlar o brilho de um led no pino PWM 10:

```

void setup()  {} // essa função pode ficar vazia se não há configuração
void loop()  {
    analogWrite(10,analogRead(A5)/4); // conversão de 0-1023 para 0-255 pela divisão por 4.
}
    
```

## B.3 - A PORTA SERIAL DO ARDUINO E SUAS FUNÇÕES EM C



**O conector USB:** É por meio desse conector USB fêmea do tipo A que o Arduino se comunica através de um cabo a um computador ou a outros dispositivos que tenham também uma interface USB. É também por esse conector que o Arduino recebe 5 volts diretamente da fonte de alimentação do computador.

Função	Exemplo	Notas
<p><b>Serial.begin(taxa)</b> Essa função habilita a porta serial e fixa a taxa de transmissão e recepção em bits por segundo entre o computador e o Arduino.</p>	<p><b>Serial.begin(9600);</b> Nesse exemplo essa função fixa a taxa de comunicação em 9600 bps. Os pinos digitais 0 e 1 não podem ser utilizados como entrada ou como saída de dados quando a porta serial é habilitada por essa função.</p>	Essa função vai sempre dentro da função <b>setup( )</b> .
<p><b>Serial.end( )</b> Desabilita a porta serial para permitir o uso dos pinos digitais 0 e 1 para entrada ou saída de dados.</p>		Essa função também deve ser escrita dentro da função <b>setup( )</b> .
<p><b>Serial.available( )</b> A função <b>Serial.available( )</b> retorna o número de bytes disponíveis para leitura no buffer da porta serial.</p>	<p><b>int total = Serial.available( );</b> Aqui a variável inteira 'total' vai guardar o número de caracteres que estão disponíveis para leitura na porta serial.</p>	O valor 0 é retornado quando não há nenhuma informação para ser resgatada na porta serial.
<p><b>Serial.read( )</b> A função <b>Serial.read( )</b> lê o primeiro byte que está no buffer da porta serial.</p>	<p><b>int valor = Serial.read( );</b> Aqui a variável inteira 'valor' vai guardar o primeiro byte (caracter) disponível na porta serial.</p>	O valor -1 é retornado quando não há nenhuma informação para ser resgatada na porta serial.
<p><b>Serial.print(valor, formato)</b> Essa função envia para a porta serial um caracter ASCII, que pode ser capturado por um terminal de comunicação. O segundo parâmetro, 'formato', é opcional e especifica com quantas casas decimais ou com que base numérica vai ser o número transmitido.</p>	<p><b>Serial.print(1.23456);</b> // transmite 1.23 (default)  <b>Serial.print(1.23456,3);</b> // transmite 1.234  <b>Serial.print("Alô Mundo!");</b> // transmite a frase (string)  <b>Serial.print('A');</b> // transmite o caracter A  <b>Serial.print('A',BIN);</b> // transmite 01000001  <b>Serial.print('A',OCT);</b> // transmite o octal 101  <b>Serial.print('A',HEX);</b> // transmite o hexa 41  <b>Serial.print('A',DEC);</b> // transmite o decimal 65</p>	
<p><b>Serial.println(valor, formato)</b> Como a anterior essa função envia para a porta serial um caracter ASCII com os mesmos parâmetros opcionais de 'formato', porem acrescenta ao final da transmissão o caracter <b>Carriage Return</b> (retorno ao início da linha) e o caracter <b>New Line</b> (mudança para a próxima linha).</p>		

Todas essas funções em C para comunicação serial podem se testadas com o código abaixo e ativando o Terminal de Comunicação do Arduino:

```
void setup( ) {
  Serial.begin(9600);           //inicia a porta serial em 9600 bps
}
void loop( ) {
  Serial.print("Retorno de 'available( )': "); //envia frase ao terminal
  Serial.println(Serial.available());         //transmite total de bytes disponíveis
  delay(1000);                                //pausa 1 seg
  Serial.print("Retorno de 'read( )': ");     //envia frase
  Serial.println(Serial.read( ));            //transmite primeiro byte disponível na porta
  delay(1000);                                //pausa 1 seg.
}
```

O Terminal Serial do Arduino mostrará inicialmente todo segundo o retorno da função **available( )**, que será 0, e o da função **read( )**, que será -1. Isso ocorre porque nenhum dado está disponível na porta serial do computador. Entre no Terminal do Arduino e transmita, por exemplo, as letras **ABCDE** juntas (digite no campo ao lado do botão 'Send') e observe que a função **available( )** informa inicialmente que existem 5 caracteres no buffer para ser enviados; e a função **read( )** mostra o código ASCII decimal 65 do primeiro deles, que corresponde à letra **A**. Os outros caracteres vão sendo enviados sequencialmente enquanto **available( )** vai decrementando até 0 de novo.

**B.4 - OS PINOS DE ALIMENTAÇÃO** Ficam na barra com 6 pinos, marcada como **POWER**, localizada ao lado dos pinos analógicos. O primeiro pino dessa barra, **RESET**, quando forçado ao potencial de terra serve para resetar o Arduino. Do outro lado, **Vin** é um pino que também pode servir para alimentar o Arduino se nele for aplicada uma tensão entre 9 e 15 volts.



**Pinos 3V3, 5V e Gnd:** dos 6 pinos dessa barra somente os quatro do meio servem para alimentar um circuito externo conectado ao Arduino: o pino de 5V e o terra (os dois pinos Gnd entre 5V e Vin); e o pino 3V3 que disponibiliza essa tensão com uma corrente máxima de 50mA.

Do mesmo autor dessa **Cartilha de Programação em C**:



Veja no site abaixo o índice e os primeiros capítulos do livro.

[www.revistadoarduino.com.br](http://www.revistadoarduino.com.br)