1. Revisão da Linguagem C

Note que a linguagem desenvolvida para a programação do arduino é baseada em C++, e deste modo mantém muita similaridade com C/C++.

Funções com sintaxe idêntica:

* while/ do-while;
  + while(condição){

<bloco de comandos>

}

executa o **bloco de comandos** enquanto a **condição** for verdadeira

* + do{

<bloco de comandos>

}while(condição)

semelhante ao anterior, com a diferença de executar **bloco de comandos pelo menos uma vez**

* for(inicio; parada; incremento);
  + itera iniciando de **início**, com um passo igual ao **incremento,** até sua variável de controle atingir o valor de **parada**
* if - else if - else;
  + If(condição){

<bloco de comandos>

}

executa o **bloco de comandos** se a **condição** for verdadeira

* + else If(condição){

<bloco de comandos>

}

executa o **bloco de comandos** se a **condição** for verdadeira, com o detalhe de apenas conferir se **condição** é verdadeira caso a **condição** do if anterior for falsa

* + else{

<bloco de comandos>

}

executa o **bloco de comandos** caso nenhuma das condições dos if’s e else if’s anteriores for verdadeira.

* criação de funções;
  + *tipo* **nome**(tipo1 parametro1, tipo2 parametro2, ... , tipoN parametroN)

1. Introdução ao Arduino

2.1 História do Arduino

O projeto Arduino teve seu início em 2005 no Interaction Design Institute, na cidade de Ivrea, na Itália. A placa foi batizada com o nome Arduino em homenagem a um bar local de mesmo nome, onde seus criadores tomavam cerveja após o expediente. Seu idealizador foi o professor Massimo Banzi, que na ocasião procurava uma solução barata para que seus estudantes de design pudessem trabalhar com tecnologia. À época, os produtos oferecidos no mercado para este fim eram demasiadamente caros e, além disso, eram relativamente difíceis de usar. Banzi compartilhou seu problema com David Cuartielles, um pesquisador visitante da Universidade de Malmö (Suécia), que estava em busca de uma solução semelhante. Juntos, eles desenvolveram um microcontrolador que pudesse ser utilizado em projetos de estudantes de design e artes. Os requisitos estabelecidos para o projeto eram o produto ser barato - a pretensão era de que o preço não poderia ser mais alto que o de uma pizza - e que qualquer pessoa fosse capaz de aprender a utilizá-lo. Assim, David Cuartielles ficou responsável desenhar a placa e David Mellis - um aluno de Massimo Banzi - programou o software para a placa. Eles ainda contrataram Gianluca Martino, um engenheiro local que também trabalhou no Design Institute, que se comprometeu a ajudar os alunos com seus projetos e a produzir um lote inicial de duzentas placas.

A placa era vendida em forma de kits para os alunos desenvolverem seus próprios projetos. Rapidamente o lote inicial foi vendido e placa ganhou popularidade, passando a ser utilizada por designers e artistas de outras regiões. Hoje em dia a placa é utilizada por hobbystas das mais diversas áreas.

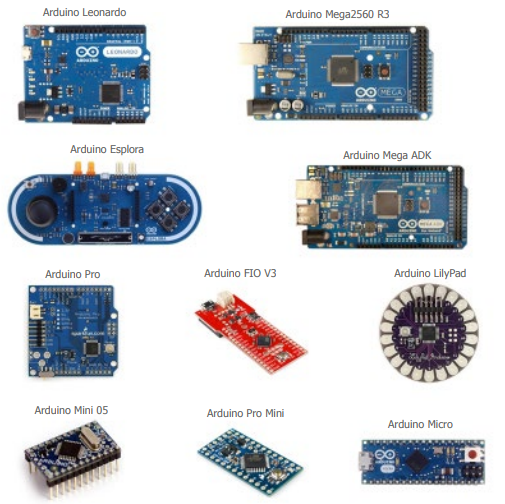
Existe uma vasta gama de versões da placa, bem como diversos clones dela. Mais adiante faremos uma comparação entre os modelos.

2.2 O que é Arduino?

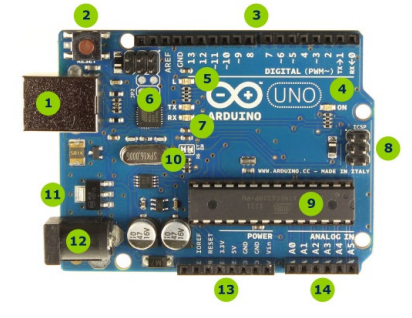
Na prática, o hardware da plataforma Arduino é um “pequeno computador”, que pode ser programado para processar sinais de entrada e saída entre ele e os componentes externos conectados a ele. O Arduino é considerado uma plataforma de computação física ou embarcada, justamente por ser um sistema capaz de interagir com o meio através de hardware e software.

3. A placa Arduino

A placa Arduino apresenta basicamente um microprocessador Atmel AVR de 8 bits, um cristal ou oscilador (relógio que envia sinais digitais em uma dada frequência) e um regulador linear de 5 volts. Assim como o modelo Uno, que será abordado neste curso, existem alguns Arduinos que possuem uma saída USB para upload do código. Além disso, todos modelos da placa dispõe de pinos digitais e analógicos (podem ser configurados como entrada ou saída de dados), pinos digitais que podem ser programados para fornecer uma saída de modulação por largura de pulso (PWM), protocolos de comunicação (serial, SPI e I2C) e um botão de reset.



Abaixo temos uma descrição um pouco mais detalhada do modelo de Arduino que será utilizado neste projeto:



1 - Conector USB

2 - Botão de reset

3 - Pinos de entrada e saída digital e PWM

4 - LED verde de placa ligada

5 - LED laranja conectado ao pino 13

6 - ATmega encarregado da comunicação com o computador

7 - LED TX (transmissor) e RX (receptor) da comunicação serial

8 - Porta ICSP para programação serial

9 - Microcontrolador ATmega 328, cérebro do Arduino

10 - Cristal de quartzo 16Mhz

11 - Regulador de tensão

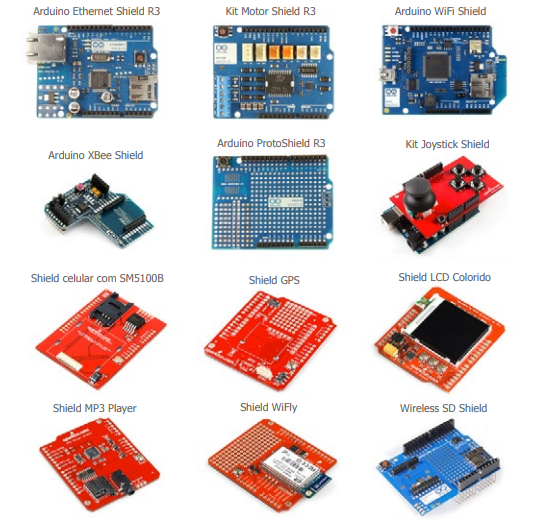
12 - Conector fêmea 2,1mm com centro positivo

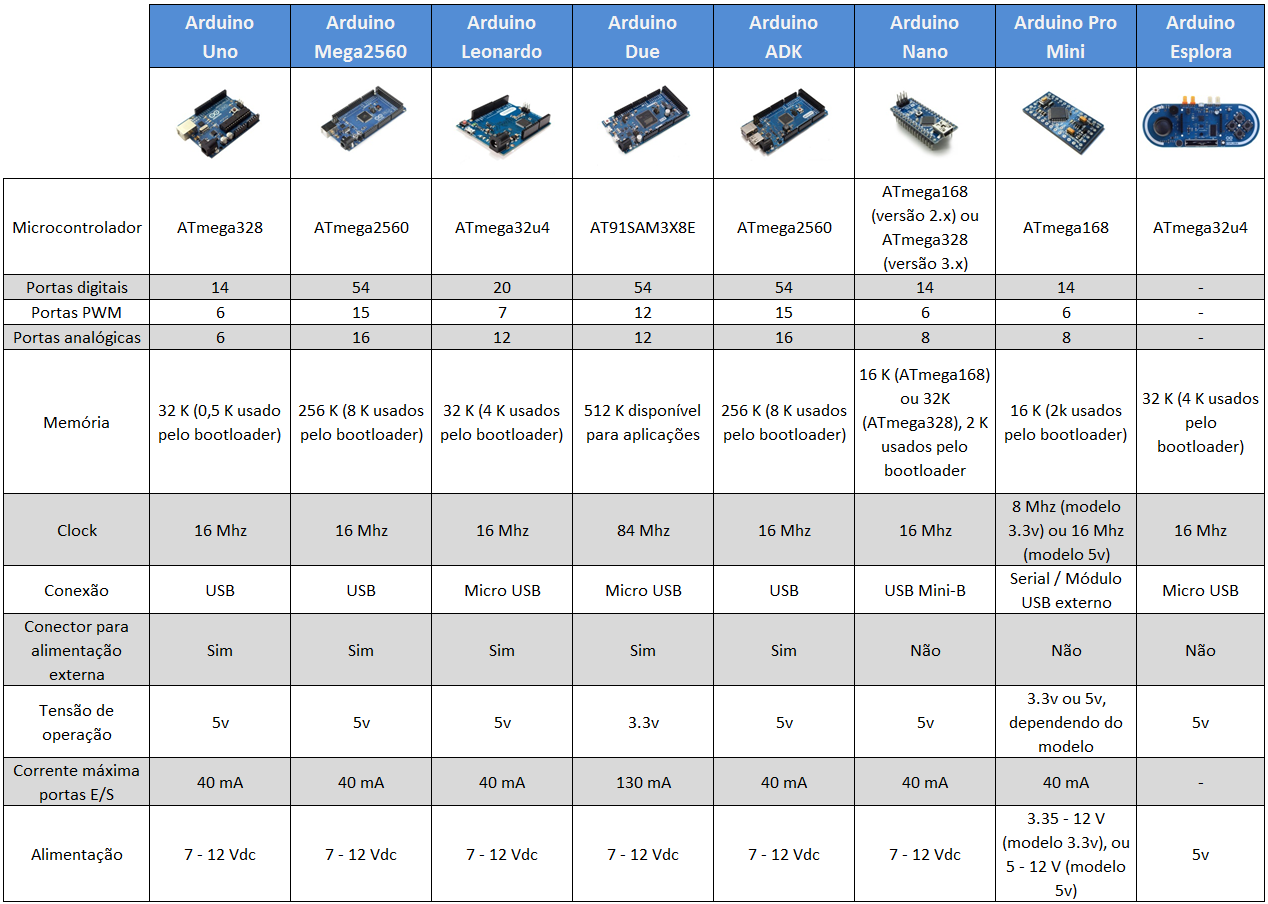
13 - Pinos de voltagem e terra

14 - Entradas analógicas

O Arduino pode ser conectados a LEDs, displays de LCD, botões, motores, sensores dos mais variados tipos (temperatura, pressão, distância, etc), entre outros. As possibilidades de projetos, considerado apenas os componentes/dispositivos citados, são inúmeras.

Apesar da diversidade de periféricos, foram desenvolvidos os chamados shields, que são placas que contém funcionalidades adicionais ao Arduino. Abaixo temos alguns exemplos de shields:

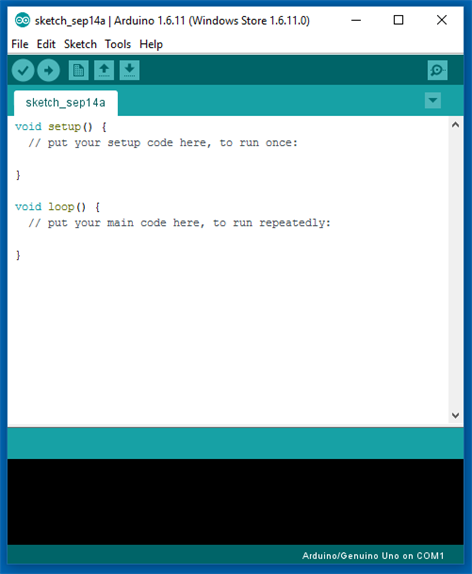
 Conforme mencionamos anteriormente, existe uma grande quantidade de modelos de Arduinos, cada qual com alterações em suas especificações, de forma a melhor atender as necessidades do usuário, tanto em termos de hardware quanto preço. Abaixo temos uma imagem de outros modelos da família Arduino e um quadro comparativo entre eles:



4. A IDE do Arduino

Quanto à programação, é utilizada uma IDE própria da plataforma, um software livre no qual é escrito o código em uma linguagem que o Arduino é capaz de interpretar, baseada em C/C++. As versões da IDE estão disponíveis para Windows, Mac OS e Linux. Certifique-se de que você fez o download da IDE para o seu respectivo sistema operacional antes da apresentação deste curso.

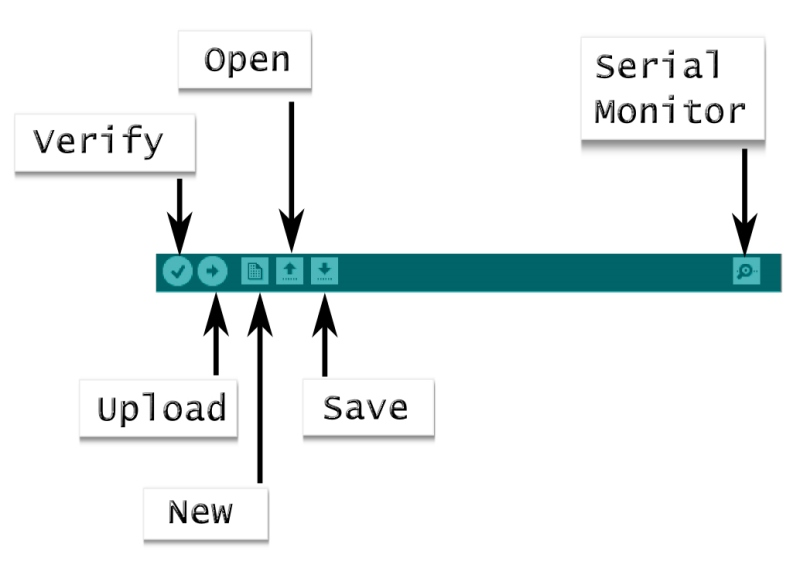
Vamos explorar um pouco a IDE do Arduino:



Quando abrimos pela primeira vez a IDE, é mostrado ao usuário um sketch em branco, como o mostrado acima. Inicialmente, é atribuído um nome temporário a este sketch. Este nome pode ser alterado para algo mais adequado quando o código for salvo.

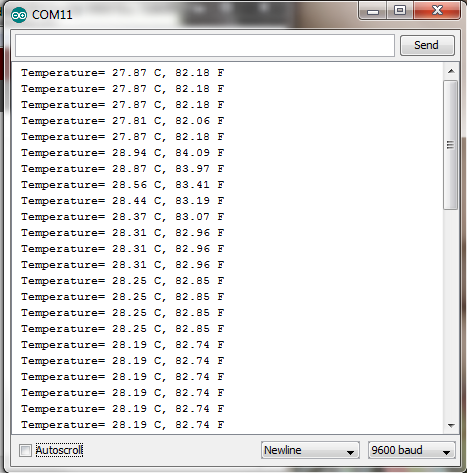
Ao observar a imagem, percebe-se que a IDE pode ser dividida em quatro partes: a barra de ferramentas na parte superior, a janela de sketch no centro e uma janela de mensagens na parte inferior, além da informação da porta serial a ser utilizada, no canto inferior direito.

Começaremos a destrinchar a barra de ferramentas. Para isto, temos em destaque os sete ícones da barra superior, com seus respectivos nomes :



|  |  |
| --- | --- |
| Verify | Verifica se há erros no código. |
| Upload | Faz o upload do sketch atual para o Arduino |
| New | Abre um novo sketch |
| Open | Exibe uma lista de sketches existentes e exemplos de sketches |
| Save | Salva o sketch atual em seu sketchbook e solicita um nome, se esse for o primeiro a ser salvo |
| Serial Monitor | Exibe os dados seriais enviados do Arduino |

O Serial monitor monitora os dados entre o Arduino e o sistema do computador através do cabo USB do Arduino. Com esta ferramenta, é possível monitorar informações armazenadas em variáveis, como por exemplo, um valor de temperatura.



Note, no canto inferior direito, a velocidade da comunicação serial (9600 baud). Em uma serial, o receptor e o transmissor devem ter a mesma velocidade, caso contrário, a informação enviada será diferente da informação recebida. Portanto, verifique se a velocidade estabelecida no código é a mesma velocidade mostrada no serial monitor, caso deseje utilizá-lo.

Na IDE existe um grande quantidade de códigos de exemplo, que auxiliam o usuário à familiarizar-se rapidamente com a linguagem.

Agora, analisaremos a parte central da IDE, onde será desenvolvido nosso código. Repare que, de antemão, já temos duas funções prontas. A primeira delas é a rotina de inicialização (setup) e a segunda é chamada de loop.

Para explicar a primeira rotina, consideremos a seguinte analogia: imagine que você deseja sair para jogar futebol. Para isto existe, uma série de coisas a se fazer antes de iniciar: calçar a chuteira, colocar roupa adequada, alongar e pegar a bola. A mesma coisa deve ser feita com o Arduino. Ele deve ser preparado ou configurado antes de ser utilizado. Estas configurações estão presentes na rotina setup, que será executada uma só vez no código. Normalmente, nesta rotina definimos a direção dos pinos de dados (entrada ou saída) e também a velocidade da transmissão da serial, entre outras coisas.

|  |
| --- |
| void setup()  {  pinMode(13, OUTPUT);  Serial.begin(9600);  } |

É importante ressaltar que, mesmo que não haja nada a ser configurado, a rotina ainda é obrigatória. Caso contrário, será gerado um erro de compilação ao tentar carregar/verificar o sketch. Neste caso, basta deixar a rotina vazia:

|  |
| --- |
| void setup()  {   } |

Quanto à rotina loop, como o próprio nome sugere, ela será executada indefinidamente, enquanto o Arduino estiver alimentado ou até alguma condição for satisfeita. Abaixo temos um código para piscar um LED:

|  |
| --- |
| void loop()  {  digitalWrite(13, HIGH);  delay(1000);  digitalWrite(13, LOW);  delay(1000);  } |

Como comentário final sobre a janela de sketch, os comentários são feitos da mesma forma que na linguagem C: ‘//’ para comentários em uma única linha e comentários entre ‘/\*\*/’ para aqueles que excedem uma linha.

Para finalizar a abordagem sobre a estrutura da IDE, comentaremos sobre a inferior da IDE, a janela de mensagens.

Como se trata de um espaço informativo, quaisquer erros que impeçam o usuário de carregar seu código para teste na placa serão exibidos nessa janela. Dentre os erros possíveis, os mais comuns são erros de compilação e o não reconhecimento do Arduino na porta COM. Os detalhes do que gerou o erro serão fornecidos, bem como a linha em que o erro ocorre (se for este o caso). Alguns erros são fáceis de identificar, outros requerem um pouco mais de experiência do usuário, pois às vezes o compilador não informa exatamente o que está errado, e essa interpretação cabe ao programador.

5. Principais funções

Abaixo temos alguns comandos importantes, bastante recorrentes em projetos utilizando Arduino:

* pinMode(pin, OUTPUT/INPUT) - este comando é utilizado na rotina de inicialização para indicar a direção (saída ou entrada) de um determinado pino analógico ou digital.
* Serial.begin(9600) - configura a comunicação serial do Arduino para utilização do serial monitor e estabelece a taxa de comunicação baud em 9600. No caso da serial do Arduino, o baud rate é o mesmo que o bitrate (taxa de bits enviados/recebidos por segundo).
* Serial.println() - comando para exibir uma mensagem ao usuário ou para verificar o valor de variáveis, ambos no monitor serial. Caso se queira exibir uma mensagem, esta deve ser colocada entre aspas duplas, se o caso for uma variável, basta colocá-la entre os parênteses
* digitalRead(pino) - lê o valor do pino digital especificado. O pino pode estar descrito como um número explicitamente (0-13), armazenado em uma variável ou escrito em uma constante.
* digitalWrite(pino, HIGH/LOW) - escreve um nível lógico alto (HIGH) ou nível lógico baixo (LOW) no pino digital escolhido.
* analogRead(pino) - lê o valor analógico do pino analógico especificado (0-5). Como a resolução do conversor A/D do Arduino é de 10 bits, o valor lido deverá estar entre a faixa de 0 a 1023.
* analogWrite(pino, valor) - escreve um valor analógico, um onda quadrada modulado por largura de pulso (PWM). Pode ser utilizado, entre outras coisas, para variar o brilho de um led ou controlar a velocidade (potência) aplicada aos terminais de um motor. Os valores possíveis para o argumento ‘valor’ são de 0 a 255. O valor zero significa que o sinal está estável em 0V por todo o período de um pulso; já o valor 255 representa um sinal estável em 5V no período do pulso. Por outro lado, o valor 64 representa um sinal quadrado em que a onda permanece em 0V por três quartos do tempo e um quarto do tempo em 5V e assim por diante. Como se trata de uma função de hardware, o pino irá gerar o sinal quadrado após a chamada à esta função e permanecerá neste estado até a próxima chamada desta função.
* delay(ms) - pausa o programa pelo tempo especificado em milissegundos. Observe que, ao chamar esta função, o programa ficará parado e só seguirá para a próxima linha quando o tempo descrito for alcançado. Portanto, tome cuidado ao usar esta função, para não prejudicar o processamento do seu programa.
* millis() - retorna o tempo decorrido (em milissegundos) entre o momento que a placa Arduino foi começou a executar o código até atingir a linha deste comando. O valor é retornado é um unsigned long (variável de 32 bits). Haverá overflow (volta a ser 0) após aproximadamente 50 dias, segundo a página do Arduino.
* randomSeed(semente) - estabelece um valor ou uma “semente” como ponto de partida para a função random(). O Arduino não é capaz de gerar números verdadeiramente aleatórios, por isso aqui se coloca uma variável ou uma função para ajudar a gerar números “um pouco mais aleatórios”. Normalmente, utiliza-se a leitura de um pino analógico sem nenhuma conexão para gerar os números. Dado que um pino sem conexão terá valores de tensão flutuantes, a cada instante teremos um valor de tensão diferente. Assim, os números gerados são “um pouco mais aleatórios”.
* randomMax(maximo) - esta função retorna números pseudo-aleatórios a partir de uma semente, conforme a explicação da função anterior. É passado como argumento da função o valor máximo que pode ser retornado.
* random(min, max)- mesma finalidade da função anterior, mas neste caso os números pseudo-aleatórios são delimitados por um valor mínimo e um valor máximo.