Resumo dos comandos do Arduino

Sketch: é o programa C que roda no Arduino.

O que é um programa?

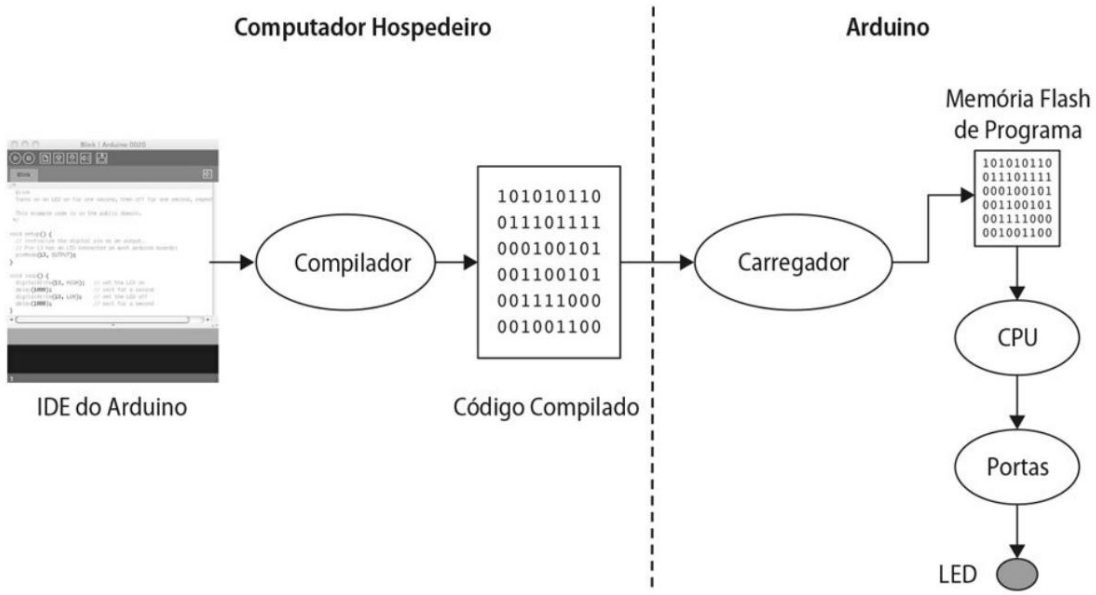
É uma lista de instruções que devem ser executadas na ordem que foram escritas.

O programa é criado usando uma linguagem de programação.

O sketch deve ser criado obedecendo a gramática da linguagem de programação (no caso C).

O sketch deve ser criado dentro de um ambiente de programação, chamado de IDE (Ambiente de Desenvolvimento Integrado).

Processo de criação do sketch e gravação do código do programa no Arduino:



Conteúdo mínimo de um sketch Arduino:

```
void setup() {
  // coloque o código que rodará uma única
  // vez aqui:
}

void loop() {
  // coloque o código que rodará repetidamente
  // (em loop) aqui:
}
```

Funções básicas e obrigatórias de todo o sketch:

```
void setup() {
        // As duas barras da data que estão no começo da linha significam que o que vem
        // depois é um comentário e não um comando em C.
        /* O que é escrito entre a "barra da data e o asterisco" , até o "asterisco e a barra da
data" é um comentário de múltiplas linhas.
        Aqui no setup, se escreve todo o código do Arduino que roda apenas uma única vez
        na inicialização do Arduino ou logo após o botão de reset ser pressionado */
}
void loop() {
        /* Aqui se coloca o código que deve ser executado constantemente ou continuamente
        (em loop). Ou seja o código que se repete*/
}
```

Definições:

Void = vazio, ou seja precede uma função que não retorna nada.

Setup() = função de inicialização que roda só uma vez na inicialização. Os parênteses em branco () significam que a função não possui valores de entrada.

Loop() = função de laço que roda continuamente

Exemplo de função que retorna algum valor:

```
sen(30) = 0.50
```

A função seno precisa de um valor de entrada, no caso o ângulo para se calcular o seno (30) e retorna um valor de saída, no caso o seno do ângulo = 0,5. Logo ela não tem **void** escrito na frente dela, pois ela retorna um valor.

Funções internas mais comuns:

Atenção! Todas as funções internas do Arduino que acabamos chamando de comandos, terminam com um ponto e virgula;

pinMode: Modo do Pino. Cada pino (GPIO - pino geral de entrada ou saída) do Arduino pode funcionar como Entrada (INPUT) ou saída (OUTPUT) dependendo da forma como for programado.

pinMode Configura um pino GPIO do Arduino como:

INPUT = entrada. ex para pegar os dados de um interruptor ou de um sensor.

OUTPUT = saída. Ex: para acender um LED ou passar dado para um Display LCD.

INPUT = entrada

OUTPUT = saída

Sintaxe:

```
  pinMode(13, OUTPUT); //configura o GPIO 13 como saída
  pinMode(ledPin, OUTPUT); //configura o GPIO armazenado na variável ledPin como saída
  pinMode(interruptor, INPUT); //configura o GPIO configurado na variável interruptor como entrada
```

onde:

pinMode: função modo do pino

13: nº do pino sendo configurado. Ou **ledPin**, nome da variável que guarda o nº do pino.

OUTPUT: modo de configuração do pino, no caso como saída.

Atenção cada comando termina com um ponto e vírgula ";"

Nomes de funções ou de variáveis criadas devem ser constituídos de apenas uma palavra e não podem ter espaços, nem acentos e nem caracteres especiais.

O que é um pino GPIO?

É um pino do Arduino que pode ser configurado (programado) para operar como uma entrada (INPUT) ou como uma saída (OUTPUT), de acordo com a necessidade do usuário.

digitalWrite: Escrita Digital. Escreve um valor lógico (digital) de zero (LOW) ou um (HIGH) na porta GPIO correspondente, ou seja , coloca a porta em nível 0 (0V) ou nível alto (3,3V ou 5V dependendo do modelo do Arduino).

LOW = baixo

HIGH = alto

Sintaxe:

```
digitalWrite(13, HIGH); //escreve 1 (5V) no pino 13

digitalWrite(13, LOW); //escreve 0 (0V) no pino 13

digitalWrite(ledPin, HIGH); //escreve 1 (5V) no nº do pino armazenado na variável ledPin

digitalWrite(ledPin, LOW); //escreve 0 (0V) no nº do pino armazenado na variável ledPin
```

onde:

digitalWrite: comando escrita digital

13: nº do pino onde se vai escrever 0 ou 1

HIGH: nível alto 3,3V ou 5V. Também chamado de nível 1

LOW: nível baixo = 0V.

ATENÇÃO, TODA O COMANDO TERMINA COM UM PONTO E VÍRGULA ;

delay: dá um atraso de tempo entre a execução do comando anterior e do comando seguinte.

Sintaxe:

```
delay(500); // o que vem entre parênteses é o tempo de atraso, sempre em ms (10-3segundos = 0,001s)
```

Exemplos:

delay(100); // espera 0,1s ou 100ms (100 * 10^{-3}s = 0,1s)

delay(500); // espera 0,5 segundos (500 * 10^{-3}s = 0,5s)

delay(1000); // espera 1s (1000 * 10^{-3}s = 1s)

delay(5000); //espera 5s (5000 * 10^{-3}s = 5s)

Funções para escrita na porta serial (console do Arduino):

```
void setup() {
Serial.begin(9600); //inicializa a interface serial do Arduino
                    // 9600 = taxa de transmissão serial em bauds/segundo.
int ledPin=13;

 Serial.println("Imprime esta linha na interface serial do Arduino.");
//Vai escrever no monitor serial:
Imprime esta linha na interface serial do Arduino.

 Serial.println(ledPin);  //imprime o valor armazenado na variável ledPin, no caso 13.
//Vai escrever no monitor serial:
13
```

Serial.println("Volume do Cilindro 1 : "+ String(vc1))

```
// Outra forma de usar a escrita serial é com o comando String, que converte o valor da variável
num tipo string (texto) que pode ser visualizado com o comando Serial.print

Supondo que o valor de vc1 = 10, vai escrever no monitor serial:
Volume do Cilindro 1 : 10
}
```

*Serial,begin deve ser configurado dentro da função de inicialização setup().

Variáveis numéricas e aritméticas:

```
int ledPin = 13;
```

Define uma variável inteira (int). ou seja números inteiros 0,1,2,3, 4,.... Também podem ser negativos como -1,-2,-3, ...

No caso acima atribuiu a variável inteira **ledPin** (pino do LED) o valor 13, ou seja, foi conectado um LED na porta 13 do Arduino e ele foi chamado de ledPin.

```
int tempoEspera = 500;
```

Define uma variável inteira (int) com nome de **tempoEspera** (tempo de espera) e se atribuiu nessa variável o valor de 500.

```
tempoEspera = tempoEspera + 100;
```

Operação de soma, soma 100 à variável **tempoEspera** e armazena-se o resultado na mesma variável.

Exemplo:

```
int delayPeriod = 100;
tempoEspera = tempoEspera + 100; //o novo valor de tempoEspera é 200
```

Formas de definição de variáveis:

```
int tempoEspera = 100; //declarei a variável tempoEspera como inteira e atribui o valor 100 a ela
```

ou

```
int tempoEspera; //apenas declarei tempoEspera como interia
tempoEspera = 100; //atribui a tempoEspera o valor 100
```

*Qualquer variável deve ser declarada somente uma única vez.

** Posso atribuir valores a uma variável diversas vezes dentro do programa.

Ex:

```
int tempoEspera; //apenas declarei tempoEspera como interia
tempoEspera = 100; //atribui a tempoEspera o valor 100
//continuo o programa ....
tempoEspera = 500; //atribui a tempoEspera o valor 500
//continuo o programa ....
tempoEspera = 3000; //atribui a tempoEspera o valor 3000
```

Comandos:

Operador condicional if (se):

Com o comando if, só executamos as funções caso o requisito avaliado seja satisfeito, ou seja caso a condição seja satisfeita (Verdadeira).

```
if (tempoEspera == 1000) {
 tempoEspera = 100;
}
```

Operador condicional if, no caso acima, compara o valor numérico armazenado na variável **tempoEspera** com o nº **1000** e se forem iguais (**==**) então executa o código entre chaves {}, no caso reatribuindo à variável **tempoEspera** o valor de **100**.

Atenção! Não confunda `tempoEspera = 100;` com `tempoEspera == 1000.`

No primeiro caso, atribuo o valor 100 à variável tempoEspera, e no segundo caso comparo se o valor armazenado na variável tempoEspera é igual (==) a 1000.

Sintaxe

```
if (condição) {
  //executa o(s) comando(s) somente se a condição definida for VERDADEIRA!
}
```

Cuidado!! Sintaxe errada não tem ponto e vírgula(;) no final e sempre tem {}!!!

```
if (delayPeriod == 1000);
  delayPeriod = 100;
```

Os operadores de comparação podem ser:

Operadores de comparação:

```
x == y (x é igual a y) //atenção são dois iguais == e não um
x != y (x é diferente de y)
x <  y (x é menor que y)
x >  y (x maior que y)
x <= y (x é menor ou igual a y)
x >= y (x é maior ou igual a y)
```

Operadores Booleanos

```
! (NÃO lógico)
&& (E lógico)
|| (OU lógico)
```

O NÃO lógico (!) resulta em verdadeiro se o operando é falso, e vice-versa. Vale lembrar que condições verdadeiras e falsas na linguagem Arduino são representadas por `true` e `false` respectivamente.

Código de Exemplo

Esse operador pode ser usado dentro da condição de um laço if.

```
if (!x) { // se x não é verdadeiro
  // código a ser executado
}
```

&&

O E lógico resulta em verdadeiro, **apenas** se ambos os operandos são verdadeiros. Vale lembrar que condições verdadeiras e falsas na linguagem Arduino são representadas por `true` e `false` respectivamente.

Código de Exemplo

Esse operador pode ser usado dentro da condição de um laço if.

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) {  // se AMBOS os botões estão em HIGH
```

```
  // código a ser executado caso as duas condições sejam verdadei-
ras
}
```

||

O OU lógico resulta em verdadeiro se pelo menos um dos operandos é verdadeiro. Vale lembrar que condições verdadeiras e falsas na linguagem Arduino são representadas por `true` e `false` respectivamente.

Código de Exemplo

Esse operador pode ser usado dentro da condição de um laço if.

```
if (x > 0 || y > 0) { // se x ou y é maior que zero
  // código a ser executado
}
```

Operador condicional for (para):

```
void loop() {
  for (int i = 0; i =< 20; i ++) {
   digitalWrite(ledPin, HIGH);
   delay(delayPeriod);
   digitalWrite(ledPin, LOW);
   delay(delayPeriod);
  }
 delay(3000);
}
```

Executa “N” repetições de uma função ou de uma sequência delas, como no caso acima (funções executados em vermelho). No caso acima executa 20 repetições das funções em vermelho.

O comando for é usado para repetir um bloco de código envolvido por chaves (bloco em vermelho

acima). Um contador de incremento (no caso é a letra "i" acima, é geralmente utilizado para terminar o loop. O loop termina no exemplo acima qdo i = 20. O comando for é útil para qualquer operação repetitiva.

Onde:

```
int i = 0; //define o valor inicial de i = 0
i =< 20; //irá executar o for até i = 20
i ++ // incrementa o valor de i depois de cada execução. Ou seja, i=0, i=1, i=2 ... i=20.
```

O for pode ter incrementos (passos) diferentes de 1 e pode ser usado com decrementos.

Exemplo para configurar todos os leds abaixo com pinMode(pinoLed, OUTPUT), de uma única vez, pode-se usar:

L8 – 38
L7 – 40
L6 – 42
L5 – 44
L4 – 46
L3 – 48
L2 – 50
L1 – 52

```
void loop() {
  for (int pinoLED = 52; pinoLED >= 38; pinoLED = pinoLED - 2) {
   pinMode(pinoLED, OUTPUT);
  }
 delay(3000);
}
```

Onde:

```
int pinoLED = 52; //define o valor inicial do pinoLED
```

pinoLED >= 38; //irá executar o for enquanto pinoLED >= 38
pinoLED = pinoLED – 2// diminui em 2 o valor do pinoLED o valor de pinoLED depois de cada interação (loop for), será: 52, 50, 48, 46, 44, 42,40 e 38.

Operador condicional while (enquanto):

```
int i = 0;
while (i < 20) //executa o while enquanto i for menor que 20
{
        //entrando no while executa as funções abaixo
        digitalwrite (ledPin, HIGH);
        delay(delayPeriod);
        digitalwrite (ledPin, LOW);
        delay(delayPeriod);
        i ++;  //incrementa i em uma unidade
}
```