Esqueleto do Código

```
// includes
// constantes
// variáveis globais
// função de setup
void setup(){
   // código que preprara o arduino
   // conforme sua necessidades
   // definindo função das portas por exemplo
   // configurações de interrupção
}

// variáveis globais usadas no loop
// e outras funções

// função loop executa sempre que preciso
void loop(){
   // código que é executado constantemente
   // deve ser construido de forma a monitorar
   // portas, tomar decisões
   // nunca deve parar, mesmo que usando
   // return será executado logo em seguida
}
```

Sintaxe

// assim é um comentário de uma linha

```
/*
 * Assim comenta-se várias linhas
 */

/*
 ou assim se preferir
 */
```

{ } as chaves demarcam blocos de código

[ ] Colchetes inicializam arrays vazios.

[x] sendo x um número inicializa arrays de x elementos

; Finaliza um alinha, deve ser usado para cada comando

#define – define uma macro/constante
// exemplo: #define CONTROLE_PORTAO

Estruturas de Controle

```
if(boolean){   } // se verdadeiro executa este bloco;
if else (boolean){   } // caso contrario verifica se este
                       // verdadeiro, se sim executa o bloco
else {   } // se nenhum for verdadeiro executa este ;

for(char i=0;i > 20;++i){   } // executa 21 vezes
                              // o bloco
        // se deseja contar inversamente,
        // troque de lugar o valor 0 com 20 e
        // e use o operador – (--i)
```

Controle Digital

```
pinMode(pino, modo); // coloca o pino no modo
                     // desejado, entrada ( INPUT)
                     // ou entrada com resistor de
                     // de pullup  ( INPUT_PULLUP)
                     // ou saída ( OUTPUT),

boolean valor = digitalRead(pino); // lê o estado
                                   // da porta
            // se ligado ( HIGH), se desligado (LOW)

digitalWrite(pino,valor); // define o estado do pino,
          // conforme valor;
          // se ligado ( HIGH),  se desligado (LOW);
```

Controle Analógico

```
analogReference(tipo); // define conversão analógica,
        //  conforme tipo parâmetros abaixo:
        // - DEFAULT: 5V padrão, ou 3.3V padrão
        // - INTERNAL: 1.1V UNO, não existe no MEGA
        // - INTERNAL1V1: 1.1V Mega somente
        // - INTERNAL2V56: 2.56V Mega somente
        // - EXTERNAL: usa como referência a tensão
        // -  do pino AREF (somente 0 a 5V )
int value = analogRead(pino); // lê  valor Analógico

analogWrite(pino,value); // escreve o valor analógico
        // na porta especificada, deve ser um
        // valor entre 0 e 255, equivale  0V até 5V
        // respectivamente, sendo que
        // 128 representa 2,5V
        // se a porta não tiver recurso PWM,
        // será escrito (LOW) para valores até de 127,
        // e (HIGH) para valores acima de 127;
```

Variáveis

char // Inteiros, ocupam 1 byte (8bits)
Valores **com** sinal, entre -128 e 127,
Valores **sem** sinal (unsigned char), entre 0 e 255
Equivalente ao tipo byte

byte // Veja "unsigned char"

int // Inteiros, ocupam 2 bytes (16bits)
// Valores entre -32768 e 32767
// ou entre 0 e 65,535 sem sinal

long // Inteiros longos, ocupam 4 bytes (32bits)
// Valores entre -2,147,483,648 e
// 2,147,483,647.
// e sem sinal entre 0 a 4,294,967,295

boolean – Equivalente a true e false,
representados por um bit,
porém ocupa um byte

float – números complexos,
representados por notação científica
Ocupam 4 bytes (32 bits)
Valores entre -3.4028235E+38 e
3.4028235E+38.

Escopo e qualificadores de variáveis

unsigned – sinaliza que uma variável não usará sinal.
signed – sinaliza que uma variável usará sinal.

const – sinaliza que é uma constante.

static – sinaliza que a variável será usada apenas
no arquivo onde foi declarada, se dentro da
função que não deve ser reinicializada;

volatile – indica que a variável será usada dentro
de uma interrupção;

#define MACRONAME value
Define uma constante do tipo macro com o nome
"MACRONAME" e valor como sendo "value"

Srings

string - um array de char;
String – um objeto do tipo Sring;

Operadores Matemáticos

= Atribuição de valor a uma variável
+ Soma dois valores ou variaveis
- Subtrai dois valroes ou variáveis

* Multiplica dois valores ou variáveis
/ Divide dois valores ou variáveis
% Modulo da divisão

>> Empurra os bits para direita
<< Empurra os bits para esquerda

++ Incremento
– – Decremento
+= Atribuição com soma
–= Atribuição com subtração
*= Atribuição com multiplicação;
/= Atribuição com divisão;
&= Atribuição usando bitwise AND
|= Atribuição usando bitwise OR

Operadores Lógicos

> testa se é valor maior que segundo valor;
(10 > 12) retorna falso;
< testa se é valor menor que segundo valor;
(10 < 12) retorna verdadeiro;
>= testa se é valor maior ou igual a segundo valor;
(15 >= 13) retorna verdadeiro;
<= testa se é valor menor ou igual a segundo valor;
(15 <= 13) retorna falso;
== testa se dois valores são iguais;
(15 == 10) retorna falso;
!= testa se dois valores são diferentes;
(15 != 10) retorna verdadeiro;
! inverte um booleano;
!(TRUE) retorna falso;

? testa um valor, se verdadeiro retorna a primeira
opção antes ":", sendo falso retorna o valor após;

&& operador "E" (and)
|| operador "OU" (or)
! (exclamação) negação

& AND bit a bit
| OR bit a bit
^ XOR bit a bit
~ not binário (inversor de bits)

Operadores Compostos

```
+=    Soma da variável com atribuição
-=    Subtrai da variável com atribuição
*=    Multiplica da variável com atribuição
/=    Subtração da variável com atribuição
+=    Soma da variável com atribuição

&=    "and" bit a bit da variável com atribuição
!=    "or" bit a bit  da variável com atribuição
```

Controle do Tempo

```
delay(tempo); // interrompe o processamento
                    // por "tempo" em milessegundos;

delayMicrosseconds(tempo); // interrompe o proc.
                    // por "tempo" em microsegundos;

millis();      // retorna total de milessegundos
               // desde a última chamada

micros();   // retorna total de microssegundos
            // desde a última chamada
```

Funções matemáticas

```
map(val1,val1min,val1max,valmin,valmax)
          // mapeia valores entre val1min e val1max
          // para valmin e valmax, para normalizar val1

min(val1, val2) //

max(val1, val2) //

abs(val1) //

constraint(val1, min, max) // delimita valor;

pow(val,p) // potência

sqrt(val) // retorna a raiz quadrada
```

Comunicação Serial

```
Serial.begin(baudrate); // Inicializa porta serial
        // Utiliza um dos baud rates padrões:
        // 300, 600, 1200, 2400, 4800, 9600 (Padrão),
        // 14400, 19200, 28800, 38400, 57600,
        // ou 115200
        // outros valores são válidos

Serial.available(); // Consulta disponibilidade
        // de bytes na porta serial;

Serial.print(val); // imprime em formato ascii na porta
Serial.println(val); // imprime em ascii com cr+lf*[1]
       /* print ou println pode receber um segundo
parâmetro informando como o valor fornecido deve ser
impresso, por exemplo "BIN" para imprimir em binário,
"HEX" para imprimir em hexadecimal, "OCT" para Octal,
"DEC" que é o padrão para decimal, ou um número para
indicar quantas casas devem ser usadas;
/*
Serial.write(val); // escreve em formato binário o valor;
Serial.writeln(""); // escreve adicionado cr+lf

Serial.read(); //

char val = Serial.peek(); // pega caracter disponível
                          // sem consumi-lo;

long value =  Serial.parseInt(); // espera um long valido;


Serial.flush(); // aguarda até que o buffer se esvazie;


void SeriialEvent(){  } // função executada ao fim
        // de cada loop em caso de chegada de
        // de dados na porta serial;

shiftIn(pino,pinoclock); // obtém o valor do pino,
                         //  sendo pinoclock o pino de pulso
shiftOut(pino,pinoclock,valor); // empurra o valor;
                         // no pino, sendo pinoclock o pulso;
```

1 cr+lf = retorno de carro mais salto de linha, conceito originado nos primórdios da computação relativo as impressoras; equivale a ASCII 10 + ASCII 13

Interrupções

interrupts() - ativa interrupções
nointerrupts() - desativa as interrupções

attachInterrupt(num, functionname, mode)
anexa a função (somente nome da função) a uma das interrupções conforme o número em parentes, o segundo número é o pino referente:
(int.0) 2 (uno, mega), (int.1)3 (uno, mega),
(int.2)21 (mega), (int.3)20 (mega),
(int.4)19 (mega), (int.5)18 (mega),
a função informada deve ter a assinatura:
void functioname() { } ;

O parâmetro "mode" deve ser:
LOW, para lançar a interrupção quando o pino tem seu sinal LOW
CHANGE, para lançar a interrupção quando o pino tem seu valor alterado
RISING, para lançar a interrupção quando o valor vai de LOW para HIGH
FALLING, quando o valor vai de HIGH para LOW

dettachInterrupt(num); desativa a interrupção;

*** ATENÇÃO *** DENTRO DE UMA INTERRUPÇÃO A FUNÇÃO DELAY(), MICROS(), MILES(), DELAYMICROSSECONDS() NÃO FUNCIONAM;

Variáveis em Memória de programa

```
#include <avr/pgmspace.h> // inclui cabeçalho
                // para trabalhar com mémoria de programa;

prog_char // tamanho de um char (1 byte)
                // valor de  -127 a 128
prog_uchar // tamanho de um char (1 byte)
                  // valores de 0 a 255
prog_int16_t // tamanho de um int (2 bytes)
                   //valores de -32,767 a 32,768
prog_uint16_t // tamanho de um int (2 bytes)
                   // valores de 0 a 65,535
prog_int32_t // tamanho de um long (4 bytes)
                   //  -2,147,483,648 to * 2,147,483,647
 prog_uint32_t //tamanho de um long (4 bytes)
                   //  0 to 4,294,967,295

// read back a 2-byte
int displayInt =
                pgm_read_word_near(charSet + k);
// read back a
char myChar =
          pgm_read_byte_near(signMessage + k);


// Then set up a table to refer to your strings.
PROGMEM const char
                        *string_table[] = {
              string_0,
              string_1,
              string_2,
              string_3,
              string_4,
              string_5
            };

for (int i = 0; i < 6; i++) {
     strcpy_P(buffer,
                  (char*)pgm_read_word(
                                &(string_table[i])));
     Serial.println( buffer );
     delay( 500 );

}
```

Digital Port Constants

HIGH
LOW
INPUT
INPUT_PULLUP
OUTPUT

Analog Port Constants

A0 até A5 Portas analogicas de 0 a 5
A6 até A Portas analógicas de 6 até, somente no Arduino Mega

Constantes de Ponto Flutuante

```
10.0        10
2.34E5      2.34 * 10^5          234000
67e-12      67.0 * 10^-12        .000000000067
```

Constantes Lógicas

TRUE	Verdadeiro
FALSE	Falso

Matemática Binária

Além dos operadores binários apresentados acima, é importante entender que.

RMB (right most bit/byte)
bit/byte de baixa ordem/menor valor
LMB (least most bit/byte)
bit/byte de alta ordem/maior valor

Constantes Numéricas Binárias, Octais e Hexadecimal

```
B00000001 <-> 1
B00000010 <-> 2
B00000011 <-> 3
...
B11111101 <-> 253
B11111110 <-> 254
B11111111 <-> 255


0x01  <->  01
0x02  <->  02
...
0x0A  <->  10
0x0B  <->  11
...
0x0F  <->  15
0x10  <->  16
0x11  <->  17
```

	Sinal Magnitude 8			(Compl. 1)	(Compl. 2)
	Binário	S/ Sinal	C/ Sinal	C/ Sinal	C/ Sinal
248	1111000	248	-122	-7	-8
249	1111001	249	-121	-6	-7
250	1111010	250	-122	-5	-6
251	1111011	251	-123	-4	-5
252	1111100	252	-124	-3	-4
253	1111101	253	-125	-2	-3
254	1111110	254	-126	-1	-2
255	1111111	255	-127	-0	-1
0	00000000	0	0	0	0
1	00000001	1	1	1	1
2	00000010	2	2	2	2
3	00000011	3	3	3	3
4	00000100	4	4	4	4
5	00000101	5	5	5	5
6	00000110	6	6	6	6
7	00000111	7	7	7	7
8	00001000	8	8	8	8

Acrónimos

Tensões no Circuito	
VCC	Tensão de alimentação positiva para o coletor, ou seja com referência ao coletor do transistor para o terra (GND), usada para indicar a tensão que alimenta circuitos eletrônicos normalmente construídos com transistores BJT do tipo NPN.
VEE	Tensão de alimentação negativa, ou seja com referencia ao emissor do transistor para o terra (GND), usada para indicar a tensão que alimenta circuitos eletrônicos normalmente construídos com transistores BJT do tipo NPN.
VDD	Como nos circuitos construídos com transistores bipolares (BJT), os circuitos baseados em transistores FET em geral tem sua tenção positiva identificada por VDD, sendo o D de dreno.
VSS	Idem ao VEE, porém para o Source dos circuitos baseados em transistores do tipo FET.
GND	Em todo circuito é preciso um referencial comum para todas as medições de tensão, as medidas do tipo VCC, VEE, VDD e VSS são feitas em relação ao terra ou seja o GND.
VCE	Medida de tensão entre o coletor e emissor de um transistor BJT
VBE	Tensão medida entre a Base e o Emissor de um transistor do tipo BJT
VC	Tensão medida em circuitos BJT, diretamente no coletor do transistor
VE	Tensão medida em circuitos BJT, diretamente no coletor do transistor
AVDD	Em circuitos que há presença de circuitos digitais e analógicos tem-se o costume de identificar a tenção que alimenta o circuito analógico pelo acrónimo AVDD, comumente usado em MCUs, também podemos encontrar como AVCC.
MCU	MicroControler Unit, Unidade MicronControladora, no caso do Arduino UNO é o chip "Atmega328"
CPU	Central Processor Unit, Unidade Central de processamento, é o coração de qualquer computador, e também de uma MCU, no caso do Arduino que usa ATmega é do tipo AVR, no Arduino DUE por exemplo que usa um núcleo do tipo ARM.
ARM	É uma arquitetura de CPU ou MCU que tem um núcleo tipicamente de 32 bits e é baseado em um conjunto de instrução do tipo RISC, hoje a arquitetura ARM permite tanto seu uso em Microcontroladores como em Microcontroladores.
AVR	
PIC	