FUNÇÕES

Para controlar a placa Arduino e realizar computações.

Entradas e Saídas Digitais

digitalRead()

digitalWrite()

pinMode()

Entradas e Saídas Analógicas

analogRead()

analogReference()

analogWrite()

Apenas Zero, Due e Família MKR

analogReadResolution()

analogWriteResolution()

Entradas e Saídas Avançadas

noTone()

pulseIn()

pulseInLong()

shiftIn()

shiftOut()

tone()

Funções Temporizadoras

delay()

delayMicroseconds()

micros()

millis()

Funções Matemáticas

abs()

constrain()

map()

max()

min()

pow()

sq()

sqrt()

Funções Trigonométricas

cos()

sin()

tan()

Caracteres

isAlpha()

isAlphaNumeric()

isAscii()

isControl()

isDigit()

isGraph()

isHexadecimalDigit()

isLowerCase()

isPrintable()

isPunct()

isSpace()

isUpperCase()

isWhitespace()

Números Aleatórios

random()

randomSeed()

Bits e Bytes

bit()

bitClear()

bitRead()

bitSet()

bitWrite()

highByte()

lowByte()

Interrupções Externas

attachInterrupt()

detachInterrupt()

Interrupções

interrupts()

noInterrupts()

Comunicação

Serial

Stream

USB

Keyboard

Mouse

VARIÁVEIS

Tipos de dados e constantes da linguagem Arduino.

Constantes

Constantes

Constantes Inteiras

Constantes de Ponto Flutuante

Conversão

byte()

char()

float()

int()

long()

word()

Tipos de Dados

String()

bool

boolean

byte

char

double

float

int

long

short

string

unsigned char

unsigned int

unsigned long

vetor

void

word

Escopo de Variáveis e Qualificadores

const

escopo

static

volatile

Utilitários

PROGMEM

sizeof()

ESTRUTURAS

Os elementos da linguagem Arduino (C++).

Sketch

loop()

setup()

Estruturas de Controle

break

continue

do...while

else

for

goto

if

return

switch...case

while

Outros Elementos da Sintaxe

#define (define)

#include (include)

/\* \*/ (comentário em bloco)

// (comentário)

; (ponto e vírgula)

{} (chaves)

Operadores Aritméticos

% (resto)

\* (multiplicação)

+ (adição)

- (subtração)

/ (divisão)

= (operador de atribuição)

Operadores de Comparação

!= (diferente de)

< (menor que)

<= (menor que ou igual a)

== (igual a)

> (maior que)

>= (maior que ou igual a)

Operadores Boleanos

! (NÃO lógico)

&& (E lógico)

|| (OU lógico)

Operadores para Ponteiros

& (referência)

\* (desreferência)

Operadores Bitwise

& (E)

<< (deslocamento à esquerda)

>> (deslocamento à direita)

^ (OU EXCLUSIVO)

| (OU)

~ (NÃO)

Operadores de Atribuição Composta

&= (atribuição por e)

\*= (atribuição por multiplicação)

++ (incremento)

+= (atribuição por adição)

-- (decremento)

/= (atribuição por divisão)

^= (atribuição por ou exclusivo)

|= (atribuição por ou)

Entradas e Saídas Digitais

digitalRead()

[Entradas e Saídas Digitais]

Descrição

Lê o valor de um pino digital especificado, que pode ser HIGH ou LOW.

Sintaxe

digitalRead(pino)

Parâmetros

pino: o número do pino digital que você quiser verificar

Retorna

HIGH ou LOW

Código de Exemplo

Aciona o pino 13 para o mesmo valor que o pino 7, declarado como entrada.

int ledPin = 13; // LED conectado ao pino digital 13

int inPin = 7; // botão conectado ao pino digital 7

int val = 0; // variável para guardar o valor lido

void setup() {

pinMode(ledPin, OUTPUT); // configura o pino digital 13 como saída

pinMode(inPin, INPUT); // configura o pino digital 7 como entrada

}

void loop() {

val = digitalRead(inPin); // lê o pino de entrada

digitalWrite(ledPin, val); // aciona o LED com o valor lido do botão

}

Notas e Advertências

Se o pino não está conectado a nada, digitalRead() pode retornar tanto HIGH como LOW (e isso pode mudar aleatoriamente).

Os pinos de entrada analógica podem ser também usados como pinos digitais, referidos como A0, A1, etc. As exceções são os pinos A6 e A7 das placas Arduino Nano, Pro Mini, e Mini, que podem ser usadas apenas como entradas analógicas.

digitalWrite()

[Entradas e Saídas Digitais]

Descrição

Aciona um valor HIGH ou LOW em um pino digital.

Se o pino for configurado como saída (OUTPUT) com a função pinMode(), sua tensão será acionada para o valor correspondente: 5V (ou 3.3V em placas alimentadas com 3.3V como o DUE) para o valor HIGH, 0V (ou ground) para LOW.

Se o pino for configurado como entrada (INPUT), a função digitalWrite() irá ativar (HIGH) ou desativar (LOW) o resistor interno de pull-up no pino de entrada. É recomendado configurar pinMode() com INPUT\_PULLUP para ativar o resistor interno de pull-up. Veja o tutorial sobre pinos digitais para mais informações.

Se você não configurar o pino com pinMode() e OUTPUT, e conectar um LED ao pino, quando chamar digitalWrite(HIGH), o LED pode aparecer um pouco apagado. Sem configurar explicitamente pinMode(), digitalWrite() irá apenas ativar o resistor de pull-up interno, que age como um grande resistor limitador de corrente.

Sintaxe

digitalWrite(pino, valor)

Parâmetros

pino: o número do pino

valor: HIGH ou LOW

Retorna

Nada

Código de Exemplo

The código configura o pino digital 13 como OUTPUT e troca seu estado entre HIGH e LOW

void setup() {

pinMode(13, OUTPUT); // configura o pino digital 13 como saída

}

void loop() {

digitalWrite(13, HIGH); // ativa o pino digital 13

delay(1000); // espera por um segundo

digitalWrite(13, LOW); // desativa o pino digital 13

delay(1000); // espera por um segundo

}

Notas e Advertências

Os pinos de entrada analógica podem ser também usados como pinos digitais, referidos como A0, A1, etc. As exceções são os pinos A6 e A7 das placas Arduino Nano, Pro Mini, e Mini, que podem ser usadas apenas como entradas analógicas.

pinMode()

[Entradas e Saídas Digitais]

Descrição

Configura o pino especificado para funcionar como uma entrada ou saída. Veja a descrição dos pinos digitais (em Inglês) para mais detalhes sobre a funcionalidade dos pinos.

Desde a versão 1.0.1, é possível ativar os resistores internos de pull-up como o modo INPUT\_PULLUP. Adicionalmente, o modo INPUT explicitamente desativa os resistores pull-up internos.

Sintaxe

pinMode(pino, modo)

Parâmetros

pino: the número do pino o qual se quer escolher o modo

modo: o modo do pino. Este pode ser INPUT, OUTPUT ou INPUT\_PULLUP; que correspondem respectivamente a entrada, saída e entrada com pull-up ativado.

Retorna

Nada

Código de Exemplo

The código configura o pino digital 13 como OUTPUT e troca seu estado entre HIGH e LOW

void setup() {

pinMode(13, OUTPUT); // configura o pino digital 13 como saída

}

void loop() {

digitalWrite(13, HIGH); // ativa o pino digital 13

delay(1000); // espera por um segundo

digitalWrite(13, LOW); // desativa o pino digital 13

delay(1000); // espera por um segundo

}

Notas e Advertências

Os pinos de entrada analógica podem ser também usados como pinos digitais, referidos como A0, A1, etc.

Entradas e Saídas Analógicas

analogRead()

[Entradas e Saídas Analógicas]

Descrição

Lê o valor de um pino analógico especificado. A placa Arduino possui um conversor analógico-digital 10 bts de 6 canais (8 canais nos Mini e Nano, 16 no Mega, 7 canais em placas MKR). Isso significa que este irá mapear tensões entre 0 e a tensão operacional (5V or 3.3V) para valores inteiros entre 0 e 1023. No Arduino UNO, por exemplo, isso permite uma resolução entre leituras de: 5 volts / 1024 unidades, ou .0049 volts (4.9 mV) por unidade. Veja a tabela abaixo para os pinos utilizáveis, tensão de operação e resolução máxima para algumas placas Arduino.

O intervalo de entrada pode ser mudado atrvés da função analogReference(), enquanto a resolução pode ser mudada (apenas nas placas Zero, Due e MKR) usando-se analogReadResolution().

Em placas baseadas em microcontroladores AVR (UNO, Nano, Mini, Mega), ler um valor analógico leva em torno de 100 microssegundos (0.0001 s), então a taxa de leitura máxima é aproximadamente 10,000 leituras por segundo.

BOARD OPERATING VOLTAGE USABLE PINS MAX RESOLUTION

Uno

5 Volts

A0 to A5

10 bits

Mini, Nano

5 Volts

A0 to A7

10 bits

Mega, Mega2560, MegaADK

5 Volts

A0 to A14

10 bits

Micro

5 Volts

A0 to A11\*

10 bits

Leonardo

5 Volts

A0 to A11\*

10 bits

Zero

3.3 Volts

A0 to A5

12 bits\*\*

Due

3.3 Volts

A0 to A11

12 bits\*\*

MKR Family boards

3.3 Volts

A0 to A6

12 bits\*\*

\*A0 through A5 are labelled on the board, A6 through A11 are respectively available on pins 4, 6, 8, 9, 10, and 12

\*\*The default analogRead() resolution for these boards is 10 bits, for compatibility. You need to use analogReadResolution() to change it to 12 bits.

Sintaxe

analogRead(pino)

Parâmetros

pino: o nome do pino de entrada analógica que se quer ler (A0 a A5 na maioria das placas, A0 a A6 em placas MKR, A0 a A7 no Mini e Nano, A0 a A15 no Mega).

Retorna

A leitura analógica no pino (int). No entanto esta é limitada a resolução do conversor analógico digital (0-1023 para 10 bits ou 0-4095 para 12 bits).

Código de Exemplo

O código abaixo lê o valor de um pino de entrada analógica e mostra seu valor na porta serial.

int analogPin = A3; // terminal do meio de um potênciometro conectado ao pino analógico 3

// terminais mais externos são conectados um no ground e o outro em +5V

int val = 0; // variável para guardar o valor lido

void setup() {

Serial.begin(9600); // configura a porta serial

}

void loop() {

val = analogRead(analogPin); // lê o pino de entrada

Serial.println(val); // imprime o valor na porta serial

}

Notas e Advertências

Se o pino de entrada analógica não estiver conectado a nada, o valor retornado por analogRead() irá flutuar baseado numa gama de fatores (ex. o valor de outros pinos analógicos, quão próxima suu mão está da placa, etc.).

analogReference()

[Entradas e Saídas Analógicas]

Descrição

Configura a tensão de referência para a entrada analógica (o valor máximo do intervalo de entrada). As opções são:

Placas Arduino AVR (Uno, Mega, etc.)

DEFAULT: a referência analógica padrão de 5 volts (em placas Arduino de 5V) ou 3.3 volts (em placas Arduino de 3.3V)

INTERNAL: uma referência interna, igual a 1.1 volts no ATmega168 ou ATmega328P e 2.56 volts no ATmega8 (Não disponível no Arduino Mega)

INTERNAL1V1: uma referência interna de 1.1V (apenas Arduino Mega)

INTERNAL2V56: uma referência interna de 2.56V (apenas Arduino Mega)

EXTERNAL: a tensão aplicada ao pino AREF (0 a 5V apenas) é usada como referência.

Placas Arduino SAMD (Zero, etc.)

AR\_DEFAULT: a referência analógica padrão de 3.3V

AR\_INTERNAL: uma referência integrada de 2.23V

AR\_INTERNAL1V0: uma referência integrada de 1.0V

AR\_INTERNAL1V65: uma referência integrada de 1.65V

AR\_INTERNAL2V23: uma referência integrada de 2.23V

AR\_EXTERNAL: a tensão aplicada ao pino AREF é usada como referência

Placas Arduino SAM (Due)

AR\_DEFAULT: a referência analógica padrão de 3.3V. Essa é a única opção suportada pelo DUE.

Sintaxe

analogReference(tipo)

Parâmetros

tipo: qual tipo de referência usar (DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56 ou EXTERNAL).

Retorna

Nada

Notas e Advertências

Após mudar-se a referência analógica, as primeiras leituras com analogRead() podem ser imprecisas.

Não use menos que 0V ou mais que 5V como tensão de referência externa no pino AREF! Se você está usando uma referência externa no pino AREF, deve configurar a referência como EXTERNAL antes de chamar analogRead(). Do contrário, você irá curto-circuitar a tensão de referência ativa (gerada internamente) e o pino AREF, possivelmente danificando o microcontrolador da sua placa Arduino.

Alternativamente, você pode conectar a tensão de referência externa ao pino AREF através de um resistor de 5K ohm, permitindo-lhe trocar entre tensões de referência externas e internas. Note que o resistor irá alterar a tensão usada como referência, pois há um resistor interno de 32K ohm no pino AREF. Os dois em conjunto agem como um divisor de tensão, então, por exemplo, 2.5V aplicados através do resistor irão fornecer 2.5 \* 32 / (32 + 5) = ~2.2V ao pino AREF.

analogWrite()

[Entradas e Saídas Analógicas]

Descrição

Aciona uma onda PWM (descrição (Em Ingês)) em um pino. Pode ser usada para variar o brilho de um LED ou acionar um motor a diversas velocidades. Após a função analogWrite() ser chamada, no pino haverá uma onda quadrada com o duty cycle (ciclo de trabalho) especificado até a próxima chamada de analogWrite() (ou uma chamada de digitalRead() ou digitalWrite() no mesmo pino). A frequência do sinal PWM na maioria dos pinos é aproximadamente 490 Hz. No Uno e placas similares, pinos 5 e 6 usam uma frequência de aproximadamente 980 Hz.

Na maioria das placas Arduino (as placas com o ATmega168 ou ATmega328), essa função funciona nos pinos 3, 5, 6, 9, 10, e 11. No Arduino Mega, essa funciona nos pinos 2 a 13 e 44 a 46. Placas mais antigas com o ATmega8 suportam analogWrite() apenas nos pinos 9, 10, and 11.

O Arduino DUE suporta analogWrite() nos pinos 2 a 13, mais pinos DAC0 e DAC1. Diferente dos pinos PWM, DAC0 e DAC1 são conversores Digital-Analógicos, e saídas analógicas legítimas.

Você não precisa chamar pinMode() para configurar um pino como saída antes de chamar analogWrite().

A função analogWrite() nada tem a ver com os pinos analógicos ou a função analogRead().

Sintaxe

analogWrite(pino, valor)

Parâmetros

pino: o pino escolhido. Tipos de dados permitidos: int.

valor: o duty cycle: entre 0 (sempre desligado) and 255 (sempre ligado). Tipos de dados permitidos: int

Retorna

Nada

Código de Exemplo

Controla a saída para um LED proporcionalmente a um valor lido de um potenciômetro.

int ledPin = 9; // LED conectado ao pino digital 9

int analogPin = 3; // potenciômetro conectado ao pino analógico 3

int val = 0; // variável para guradar o valor lido

void setup() {

pinMode(ledPin, OUTPUT); // configura o pino como saída

}

void loop() {

val = analogRead(analogPin); // lê o pino de entrada analógica

analogWrite(ledPin, val / 4); // analogRead retorna valores de 0 a 1023, analogWrite recebe de 0 a 255

}

Notas e Advertências

As ondas PWM generadas nos pinos 5 e 6 terão duty cycles maiores que o esperado. Isso acontece porque ocorrem interações com as funções millis() e delay(), as quais compartilham o mesmo timer (temporizador) interno usado para gerar essas saídas PWM. Isso será constatado principalmente em duty-cycle baixos (ex. 0 - 10) e podem resultar em um valor de 0 não desligando completamente a saída nos pinos 5 e 6.

Apenas Zero, Due e Família MKR

analogReadResolution()

[Apenas Zero, Due e Família MKR]

Descrição

analogReadResolution() é um extensão da API Analog para o Arduino Due, Zero e família MKR.

Configura o tamanho (em bits) do valor retornado por analogRead(). O padrão é 10 bits (retorna valores entre 0-1023) para compatibilidade com placas baseadas em microcontroladores AVR.

As placas Due, Zero e da família MKR possuem um conversor analógico digital (ADC) com capacidade 12-bits que pode ser accessado, em completude, mudando-se a resolução para 12. Isso fará com que os valores retornados por analogRead() estejam entre 0 e 4095.

Sintaxe

analogReadResolution(bits)

Parâmetros

bits: determina a resolução (em bits) dos valores retornados pela função analogRead(). Você pode configurar esse valor entre 1 e 32. Você pode escolher resoluções mais altas que 12, porém assim, valores retornados por analogRead() irão sofrer aproximação. Veja a nota abaixo para detalhes.

Retorna

Nada

Código de Exemplo

O código abaixo mostra como usar o conversor analógico digital com resoluções diferentes.

void setup() {

// abre a conexão serial

Serial.begin(9600);

}

void loop() {

// lê o valor no pino A0 na resolução padrão (10 bits)

// e o envia pela conexão serial

analogReadResolution(10);

Serial.print("ADC 10-bit (padrão) : ");

Serial.print(analogRead(A0));

// muda a resolução para 12 bits e lê o pino A0

analogReadResolution(12);

Serial.print(", 12-bit : ");

Serial.print(analogRead(A0));

// muda a resolução para 16 bits e lê o pino A0

analogReadResolution(16);

Serial.print(", 16-bit : ");

Serial.print(analogRead(A0));

// muda a resolução para 8 bits e lê o pino

analogReadResolution(8);

Serial.print(", 8-bit : ");

Serial.println(analogRead(A0));

// um pequeno delay para não enviar dados muito rapidamente para o Serial Monitor

delay(100);

}

Notas e Advertências

Se você configurar o valor de analogReadResolution() para um valor maior que acapacidade de sua placa, o Arduino irá retornar apenas na sua resolução máxima, preenchendo os bits extras com zeros.

Por exemplo: usando-se o DUE com analogReadResolution(16) irá retornar um número 16-bit aproximado, com os primerios 12 bits contendo a leitura real do ADC e os últimos 4 bits preenchidos com zeros.

Se você configurar o valor de analogReadResolution() para um valor menor que acapacidade de sua placa, os bits menos significantes extras lidos do ADC serão descartados.

Usar uma resolução de 16 bits (ou qualquer resolução mais alta que as capacidades reais do hardware) permite esccrever sketches que automaticamente suportam dispositivos com uma resolução do ADC mais alta quando estes estiverem disponíveis em palcas futuras, sem a necessidade de se fazer mudanças no código.

analogWriteResolution()

[Apenas Zero, Due e Família MKR]

Descrição

analogWriteResolution() é uma extensão da API Analog para os Arduinos Due, Zero e MKR.

analogWriteResolution() configura a resolução da função analogWrite(). O padrão é 8 bits (valores entre 0-255) para compatibilidade com placas baseadas em microcontroladores AVR.

O Due possui as seguintes capacidades em seu hardware:

12 pinos com PWM de 8-bit por padrão, como as placas baseadas em AVR. Estes podem ser mudados para resolução 12-bits.

2 pinos com DAC 12-bit (Do Inglês, Digital-to-Analog Converter).

Ao configurar a resolução para 12 bits, você pode usar analogWrite() com valores entre 0 e 4095 para tirar vantagem da resolução completa do DAC ou do sinal PWM.

O Zero possui as seguintes capacidades em seu hardware:

10 pinos com PWM de 8-bit por padrão, como as placas baseadas em AVR. Estes podem ser mudados para resolução 12-bits.

1 pinos com DAC 12-bit (Do Inglês, Digital-to-Analog Converter).

Ao configurar a resolução para 10 bits, você pode usar analogWrite() com valores entre 0 e 1023 para tirar vantagem da resolução completa do DAC.

A família MKR de placas possui as seguintes capacidades em seu hardware:

4 pinos com PWM de 8-bit por padrão, como as placas baseadas em AVR. Estes podem ser mudados para resolução 12-bits.

1 pino com DAC 10-bit (Do Inglês, Digital-to-Analog Converter).

Ao configurar a resolução para 12 bits, você pode usar analogWrite() com valores entre 0 e 4095 para sinais PWM; Use 10 bits no pino do conversor digital-analógico para tira vantagem da resoulção completa de 1024 valores do DAC.

Sintaxe

analogWriteResolution(bits)

Parâmetros

bits: determina a resolução (em bits) dos valores usados na função analogWrite(). Se você escolher uma resolução maior ou menor que a capacidade do hardware da sua placa, o valor usado em analogWrite() será ou truncado se é muito grande ou completo com preenchido com zeros se for muito pequeno. Veja a nota abaixo para detalhes.

Retorna

Nada

Código de Exemplo

O código configura a resolução da saída PWM para diferentes valores.

void setup() {

// inicia a porta serial

Serial.begin(9600);

// configura os pinos digitais 11 a 13 como saída

pinMode(11, OUTPUT);

pinMode(12, OUTPUT);

pinMode(13, OUTPUT);

}

void loop() {

// Lê o valor analógico em A0 e mapeia-o para

// um pino PWM com um LED conectado

int sensorVal = analogRead(A0);

Serial.print("Analog Read) : ");

Serial.print(sensorVal);

// a resolução PWM padrão

analogWriteResolution(8);

analogWrite(11, map(sensorVal, 0, 1023, 0, 255));

Serial.print(" , 8-bit PWM value : ");

Serial.print(map(sensorVal, 0, 1023, 0, 255));

// Muda a reolução PWM para 12 bits

// A resolução completa de 12 bits é suportada

// apenas no Due

analogWriteResolution(12);

analogWrite(12, map(sensorVal, 0, 1023, 0, 4095));

Serial.print(" , 12-bit PWM value : ");

Serial.print(map(sensorVal, 0, 1023, 0, 4095));

// Muda a reolução PWM para 4 bits

analogWriteResolution(4);

analogWrite(13, map(sensorVal, 0, 1023, 0, 15));

Serial.print(", 4-bit PWM value : ");

Serial.println(map(sensorVal, 0, 1023, 0, 15));

delay(5);

}

Notas e Advertências

Se você configurar analogWriteResolution() com um valor maior que a capacidade de sua placa, o Arduino irá descartar os bits extras. Por exemplo: ao configurar o Due com analogWriteResolution(16) em um pino DAC 12-bit, apenas os primeiros 12 bits dos valores passados para analogWrite() serão usados e os outros 4 bits serão descartados.

Se você configurar analogWriteResolution() com um valor menor que a capacidade de sua placa, os bits ausentes serão completos com zeros para preencher o tamanho requerido pelo hardware. Por exemplo: ao configurar o Due com analogWriteResolution(8) em um pino DAC 12-bit, o Arduino irá adicionar 4 bits zero a esquerda do valor 8-bit usado em analogWrite() para obter os 12 bits requeridos.

Entradas e Saídas Avançadas

noTone()

[Entradas e Saídas Avançadas]

Descrição

Interrompe a geração de uma onda quadrada iniciada pela função tone(). Não tem nenhum efeito se nenhum tom está sendo gerado.

Sintaxe

noTone(pino)

Parâmetros

pino: o pino no qual se quer parar de gerar um tom.

Retorna

Nada

Notas e Advertências

Se você quiser tocar tons diferentes em múltiplos pinos, você precisa chamar noTone() em um pino antes de chamar tone() no próximo pino.

pulseIn()

[Entradas e Saídas Avançadas]

Descrição

Captura a duração de um pulso em um pino (que pode ser HIGH ou LOW). Por exemplo, se o valor HIGH é passado para a função, a função pulseIn() espera o pino ir para do estado 'LOW' para HIGH, começa a temporizar, então espera o pino ir para o estado LOW e para de temporizar. Retorna o tamanho do pulso em microssegundos ou desiste e retorna 0 se não receber nenhum pulso dentro de um tempo máximo de espera especificado.

A temporização dessa função foi determinada empiricamente e irá provavelmente mostrar erros em pulsos mais longos. funciona em pulsos de 10 microssegundos a 3 minutos de duração.

Sintaxe

pulseIn(pino, valor)

pulseIn(pino, valor, tempo\_de\_espera)

Parâmetros

pino: o número do pino no qual se quer capturar a duração de um pulso. (int)

valor: tipo de pulso a ser lido: pode ser HIGH ou LOW. (int)

tempo\_de\_espera (opcional): o número de microssegundos a se esperar pelo começo do pulso; o padrão é um segundo. (unsigned long)

Retorna

A duração do pulso (em microssegundos) ou 0 se nenhum pulso começar antes de se esgotar o tempo de espera (unsigned long).

Código de Exemplo

O exemplo abaixo imprime a duração de um pulso no pino 7.

int pino = 7; //pino para a entrada do pulso

unsigned long duracao; //variável para guardar a duração do pulso

void setup()

{

Serial.begin(9600);

pinMode(pino, INPUT);

}

void loop()

{

duracao = pulseIn(pino, HIGH);

Serial.println(duracao);

}

pulseInLong()

[Entradas e Saídas Avançadas]

Descrição

pulseInLong() é uma alternativa à função pulseIn(), sendo melhor para lidar com pulsos longos e situações afetadas por interrupções.

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseInLong() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or 0 if no complete pulse was received within the timeout.

A temporização dessa função foi determinada empiricamente e irá provavelmente mostrar erros em pulsos mais curtos. Funciona em pulsos de 10 microssegundos a 3 minutos de duração. Por favor note que se a função for configurada para recebr um pulso HIGH e o pino já estiver no estado HIGH quando a função for chamada, essa irá primeiro esperar o pino ir para estado LOW e então HIGH antes de começar a contar. Essa função pode ser utilizada apenas quando interrupções estiverem ativadas. Além disso, resoluções maiores são alcançadas com intervalos maiores.

Sintaxe

pulseInLong(pino, valor)

pulseInLong(pino, valor, tempo\_de\_espera)

Parâmetros

pino: o número do pino no qual se quer capturar a duração de um pulso. (int)

valor: tipo de pulso a ser lido: pode ser HIGH ou LOW. (int)

tempo\_de\_espera (opcional): o número de microssegundos a se esperar pelo começo do pulso; o padrão é um segundo. (unsigned long)

Retorna

A duração do pulso (em microssegundos) ou 0 se nenhum pulso começar antes de se esgotar o tempo de espera (unsigned long).

Código de Exemplo

O exemplo abaixo imprime na porta serial a duração de um pulso no pino 7.

int pino = 7; // pino para a entrada do pulso

unsigned long duracao; // variável para guardar a duração do pulso

void setup() {

Serial.begin(9600);

pinMode(pino, INPUT);

}

void loop() {

duracao = pulseInLong(pino, HIGH);

Serial.println(duracao);

}

Notes and Warnings

Essa função depende da função micros(), então não pode ser usada quando interrupções estiverem desativadas com noInterrupts().

shiftIn()

[Entradas e Saídas Avançadas]

Descrição

Recebe um byte de dados, um bit de cada vez. Começa com ou o bit mais significante (o mais à esquerda) ou o menos significante (mais à direita). Para cada bit, o pino de clock é colocado em estado em HIGH, o próximo bit é lido no pino de dados (data), e então o pino é colocado em estado LOW novamente.

Se está conectando um dispositivo que funciona a partir da borda de subida sinal de clock, você irá precisar garantir que o pino de clock está em estado LOW antes da primeira chamada de shiftIn(), ex. chamando digitalWrite(clockPin, LOW).

Nota: essa é uma implementação por software; O Arduino também provê uma biblioteca SPI que faz a implementação em hardware, que é mais rápida, mas apenas funciona em pinos específicos.

Sintaxe

byte incoming = shiftIn(dataPin, clockPin, bitOrder)

Parâmetros

dataPin: o pino no qual receber cada bit (int).

clockPin: o pino alternado para sinalizar uma leitura de dataPin.

bitOrder: em que ordem receber os bits; pode ser MSBFIRST ou LSBFIRST. Correspondendo respectivamente a primeiro o bit mais significativo (Most Significant Bit First), ou o primeiro o bit menos significativo (Least Significant Bit First)

Retorna

O valor lido (byte).

shiftOut()

[Entradas e Saídas Avançadas]

Descrição

Transfere um byte de dados um bit de cada vez. Começa com ou o bit mais significante (o mais à esquerda) ou o menos significante (mais à direita). Cada bit é escrito em sequência em um pino data, logo após o pino clock é pulsado (colocado em HIGH, depois LOW) para indicar que aquele bit está disponível.

Nota - se você está conectando um dispositivo que é sincronizado pela borda de subida do clock, irá precisar fazer com que o pino clock seja low antes de chamar shiftOut(), ex. com digitalWrite(clockPin, LOW).

Essa é uma implementação por software; O Arduino também provê uma biblioteca SPI que faz a implementação em hardware, que é mais rápida, mas apenas funciona em pinos específicos.

Sintaxe

shiftOut(dataPin, clockPin, bitOrder, value)

Parâmetros

dataPin: o pino no qual transferir cada bit (int)

clockPin: o pino a ser pulsado uma vez que o pino data estiver com o bit a ser trasnferido (int)

bitOrder: em que ordem receber os bits; pode ser MSBFIRST ou LSBFIRST. Respectivamente, primeiro o bit mais significativo (Most Significant Bit First), ou o primeiro o bit menos significativo (Least Significant Bit First)

value: o valor a ser transferido. (byte)

Retorna

Nada

Código de Exemplo

Para o circuito, ver o tutorial sobre o controle de um registrador de deslocamento 74HC595 (Em Inglês).

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Name : shiftOutCode, Hello World //

// Author : Carlyn Maw,Tom Igoe //

// Date : 25 Oct, 2006 //

// Version : 1.0 //

// Notes : Code for using a 74HC595 Shift Register //

// : to count from 0 to 255 //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Pino conectado a ST\_CP no 74HC595

int latchPin = 8;

//Pino conectado a SH\_CP no 74HC595

int clockPin = 12;

//Pino conectado a DS no 74HC595

int dataPin = 11;

void setup() {

//configura os pinos usados no loop principal como saídas

pinMode(latchPin, OUTPUT);

pinMode(clockPin, OUTPUT);

pinMode(dataPin, OUTPUT);

}

void loop() {

//rotina de contagem de 0 até 255

for (int j = 0; j < 256; j++) {

//coloca e mantém o pino latch em low enquanto ocorre a transmissão

digitalWrite(latchPin, LOW);

//transmite o valor de j, a começar pelo bit menos significativo

shiftOut(dataPin, clockPin, LSBFIRST, j);

//retorna o pino latch para high para sinalizar ao chip

//que esse não precisa mais esperar por informação

digitalWrite(latchPin, HIGH);

delay(1000);

}

}

Notas e Advertências

Os pinos data e clock devem ser configurados como saídas com uma chamada de pinMode().

A função shiftOut() atualmente funciona para transferir apenas 1 byte (8 bits) então requer uma operação em dois passos para transferir valores maiores que 255.

// Para serial MSBFIRST, faça:

int data = 500;

// transfere o byte mais significativo

shiftOut(dataPin, clock, MSBFIRST, (data >> 8));

// trasnfere o byte menos significativo

shiftOut(dataPin, clock, MSBFIRST, data);

// Para serial LSBFIRST, faça:

data = 500;

// transfere o byte menos significativo

shiftOut(dataPin, clock, LSBFIRST, data);

// transfere o byte mais significativo

shiftOut(dataPin, clock, LSBFIRST, (data >> 8));

tone()

[Entradas e Saídas Avançadas]

Descrição

Gera uma onda quadrada na frequência especificada (e duty cycle 50%) em um pino. A duração pode ser especificada, do contrário a onda continua até uma chamada de noTone(). O pino pode ser conectado a um buzzer piezo ou outro speaker para tocar tons.

Apenas um tom pode ser gerado de cada vez. Se um tom já está tocando em um pino diferente, a chamada de tone() não terá efeito. Se o tom está tocando no mesmo pino, a chamada irá mudar sua frequência para a nova especificada.

Uso da função tone() irá interferir com saída PWM nos pinos 3 e 11 (em placas diferentes do Mega).

Não é possível gerar tons de freqeuência mais baixa que 31Hz. Para detalhes técnicos, veja as notas de Brett Hagman’s(Em Inglês).

Sintaxe

tone(pino, frequência)

tone(pino, frequência, duração)

Parâmetros

pino: o pin no qual gerar o tom

frequência: a frequência do tom em Hertz - unsigned int

duração: a duração do tom em milissegundos (opcional) - unsigned long

Retorna

Nada.

Notas e Advertências

Se você quiser tocar tons diferentes em múltiplos pinos, você precisa chamar noTone() em um pino antes de chamar tone() no próximo pino.

Funções Temporizadoras

delay()

[Funções Temporizadoras]

Descrição

Pausa o programa por uma quantidade especificada de tempo (em milissegundos). Cada segundo equivale a 1000 milissegundos.

Sintaxe

delay(ms)

Parâmetros

ms: o número de milissegundos para pausar o programa (unsigned long)

Retorna

Nada

Código de Exemplo

O código pausa o programa por um segundo antes de trocar o estado do pino 13.

int ledPin = 13; // LED conectado ao pino digital 13

void setup() {

pinMode(ledPin, OUTPUT); // configura o pino digital como saída

}

void loop() {

digitalWrite(ledPin, HIGH); // acende o LED

delay(1000); // espera por um segundo

digitalWrite(ledPin, LOW); // apaga o LED

delay(1000); // espera por um segundo

}

Notas e Advertências

Mesmo que seja fácil fazer um LED piscar usando a função delay(), e muitos sketches usam delays pequenos para tarefas como debouncing de botões, o uso de delay() em um sketch possui desvantagens significantes. Nenhuma leitura de sensores, cálculos matemáticos, ou manipulação de pinos podem ocorrer durante a função delay(), para resumir, causa a pausa de qualquer atividade. Para métodos alternativos de controlar temporizações, veja o sketch Blink Sem Delay (Em Inglês), que verifica a função millis() até que o tempo suficiente tenha passado. Programadores mais habilidosos usualmente evitam o uso da função delay() para timing de eventos mais longos que dezenas de milissegundos, a menos que o sketch Arduino seja muito simples.

No entanto, certas coisas continuam a acontecer enquanto a função delay() está controlando o microcontrolador, porque a função delay não desativa interrupções. Comunicação serial recebida no pino RX é armazenada, valores PWM de (analogWrite) e estados dos pinos são mantidos, e interrupções externas irão funcionar como devem.

delayMicroseconds()

[Funções Temporizadoras]

Descrição

Pausa o programa pela quantidade de tempo especificada como parâmetro (em microssegundos). Há mil microssegundos em um milissegundo, e um milhão de microssegundos em um segundo.

Atualmente, o maior valor que irá porduzir um delay preciso é 16383. Isso pode mudar em versões futuras do Arduino. Para delays mais longos que alguns milhares de microssegundos, você deve usar delay() em vez disso.

Sintaxe

delayMicroseconds(us)

Parâmetros

us: o número emm microssegundos para pausar o programa (unsigned int)

Retorna

Nada

Código de Exemplo

O código abaixo configura o pino 8 para funcionar como pino de saída. Ele então gera um trem de pulsos com período de 100 microssegundos.

int outPin = 8; // pino digital 8

void setup() {

pinMode(outPin, OUTPUT); // configura o pino digital como saída

}

void loop() {

digitalWrite(outPin, HIGH); // ativa o pino

delayMicroseconds(50); // pausa por 50 microssegundos

digitalWrite(outPin, LOW); // desativa o pino

delayMicroseconds(50); // pausa por 50 microssegundos

}

Notas e Advertências

Essa função funciona bastante precisamente para valores maiores que 3 microssegundos. Não é possível assegurar que delayMicroseconds() irá funcionar corretamente para valores menores.

A partir da versão Arduino 0018, delayMicroseconds() não mais desativa interrupções.

micros()

[Funções Temporizadoras]

Descrição

Retorna o número de microssegundos passados desde que a placa Arduino começou a executar o programa atual. Esse número irá sofrer overflow (chegar ao maior número possível e então voltar pra zero), após aproximadamente 70 minutos. Em placas Arduino 16 MHz (ex. UNO e Nano), essa função possui uma resolução de quatro microssegundos (isto é, o número retornado é sempre um múltiplo de quatro). Em placas Arduino 8 MHz (ex. LilyPad), essa função possui uma resolução de oito microssegundos.

Sintaxe

time = micros()

Parâmetros

Nenhum

Retorna

O número de microssegundos desde que o programa iniciou (unsigned long).

Código de Exemplo

O código abaixo imprime o número de microssegundos passados desde que a placa Arduino foi ligada.

unsigned long time;

void setup() {

Serial.begin(9600);

}

void loop() {

Serial.print("Time: ");

time = micros();

Serial.println(time); // imprime o tempo desde que o programa iniciou

delay(1000); // espera um segundo, para não enviar quantidades massivas de dados

}

Notas e Advertências

Há 1000 (mil) microssegundos em um milissegundo e 1000000 (um milhão) de microssegundos em um segundo.

millis()

[Funções Temporizadoras]

Descrição

Retorna o número de milissegundos passados desde que a placa Arduino começou a executar o programa atual. Esse número irá sofrer overflow (chegar ao maior número possível e então voltar pra zero), após aproximadamente 50 dias.

Sintaxe

time = millis()

Parâmetros

Nenhum

Retorna

O número de milissegundos passados desde que o programa iniciou (unsigned long)

Código de Exemplo

O código imprime na porta serial o tempo em milissegundos passado desde que a placa Arduino começou a executar o código em si.

unsigned long time;

void setup() {

Serial.begin(9600);

}

void loop() {

Serial.print("Time: ");

time = millis();

Serial.println(time); // imprime o tempo desde que o programa iniciou

delay(1000); // espera um segundo, para não enviar quantidades massivas de dados

}

Notas e Advertências

Note que o valor retornado por millis() é unsigned long, erros podem ser gerados se o programador tentar fazer operações matemáticas com outros tipos de dados, como int. Até mesmo o tipo long com sinal pode causar erros, já que seu valor máximo é metade de sua contraparte sem sinal.

Funções Matemáticas

abs()

[Funções Matemáticas]

Descrição

Calcula o módulo (ou valor absoluto) de um número.

Sintaxe

abs(x)

Parâmetros

x: o número do qual se deseja saber o módulo

Retorna

x: se x é maior ou igual a 0.

-x: se x é menor que 0.

Notas e Advertências

Por causa da forma em que a função abs() é implementada, evite usar outras funções dentro dos parênteses, isso pode levar a resultados incorretos.

abs(a++); // evitar isso - causa resultados incorretos

abs(a); // Ao invés disso, usar esta forma

a++; // manter a aritmética fora da função

constrain()

[Funções Matemáticas]

Descrição

Restringe um número a ficar dentro de um intervalo.

Sintaxe

constrain(x, a, b)

Parâmetros

x: o número a se restringir, qualquer tipo de dado

a: o extremo inferior do intervalo, qualquer tipo de dado

b: o extremo superior do intervalo, qualquer tipo de dado

Retorna

x: se x está entre a e b

a: se x é menor que a

b: se x é maior que b

Código de Exemplo

O código limita os valores de uma variável com a leitura de um sensor entre 10 e 150.

sensVal = constrain(sensVal, 10, 150); // limita os valores entre 10 e 150

Notas e Advertências

Por causa da forma como a função constrain() é implementada, evite usar outras funções dentro dos parênteses, pois isso pode levar a resultados incorretos.

Esse código gerará resultados incorretos:

int constrainedInput = constrain(Serial.parseInt(), minimumValue, maximumValue); // evite isso

Faça essa forma:

int input = Serial.parseInt(); // manter outras operações fora da função constrain()

int constrainedInput = constrain(input, minimumValue, maximumValue);

map()

[Funções Matemáticas]

Descrição

Remapeia um número de um intervalo para outro. Isto é, um valor de deMenor seria mapeado para paraMenor, um valor de deMaior para paraMaior, valores dentro de uma faixa para volores dentro da outra faixa, etc.

Não restringe valores a ficar dentro do intervalo, porque valores fora do intervalo são as vezes úteis e pretendidos. A função constrain() pode ser usada tanto antes como depois dessa função, se limites para os intervalos são desejados.

Note que os "limites mínimos" de cada intervalo podem ser maiores ou menores que os "limites máximos" tal que a função map() pode ser usada para reverter um intervalo de números, por exemplo

y = map(x, 1, 50, 50, 1);

A função também funciona bem com números negativos, tal que esse exemplo

y = map(x, 1, 50, 50, -100);

também é valido e funciona bem.

A função map() usa números inteiros e não irá gerar números fracionários, quando a matemática indicar que deveria. Resíduos fracionários são truncados e não são arredondados.

Sintaxe

=map(valor, deMenor, deMaior, paraMenor, paraMaior);

Parâmetros

valor: o número a ser mapeado

deMenor: o menor limite do intervalo atual do valor

deMaior: o maior limite do intervalo atual do valor

paraMenor: o menor limite do intervalo alvo

paraMaior: o maior limite do intervalo alvo

Retorna

O valor mapeado para o novo intervalo.

Código de Exemplo

/\* Mapeia um valor analógico para 8 bits (0 a 255) \*/

void setup() {}

void loop() {

int val = analogRead(0);

val = map(val, 0, 1023, 0, 255);

analogWrite(9, val);

}

Apêndice

Para os matematicamnete interessados, aqui tem-se o código da função:

long map(long x, long in\_min, long in\_max, long out\_min, long out\_max) {

return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min;

}

Notes & Warnings

Como mencionado anteriormente, a função map() usa matemática inteira. Então frações podem ser suprimidas devido a isso. Por exemplo, frações como 3/2, 4/3, 5/4 irão ser retornadas como 1 pela função map(), apesar de os valores esperados serem diferentes. Então, se seu projeto requer cálculos precisos (ex. tensão elétrica com precisão de três decimais), por gentileza considere evitar a função map() e implementar os cálculos manualmente no seu código.

max()

[Funções Matemáticas]

Descrição

Calcula o maior de dois números.

Sintaxe

max(x, y)

Parâmetros

x: o primeiro número, qualquer tipo de dado

y: o segundo número, qualquer tipo de dado

Retorna

O maior dos dois números passados para a função.

Código de Exemplo

O código garante que o valor de sensVal seja pelo menos 20.

sensVal = max(sensVal, 20); // atribui a sensVal o maior valor, seja sensVal ou 20

// (efetivamente garantindo que sensVal seja ao menos 20)

Notas e Advertências

Talvez contraintuitivamente, max() é constantemente usada para restringir o extremo inferior do intervalo de uma variável, enquanto min() é usado para restringir o extremo superior do intervalo.

Por causa da forma em que a função max() é implementada, evite usar outras funções dentro dos parênteses, isso pode levar a resultados incorretos.

max(a--, 0); // evitar isso - causa resultados incorretos

max(a, 0); // ao invés disso, usar esta forma

a--; // manter a aritmética fora da função

min()

[Funções Matemáticas]

Descrição

Calcula o menor de dois números.

Sintaxe

min(x, y)

Parâmetros

x: o primeiro número, qualquer tipo de dado

y: o segundo número, qualquer tipo de dado

Retorna

O menor dos dois números passados para a função.

ECódigo de Exemplo

O código garante que o valor de sensVal nunca é maior que 100.

sensVal = min(sensVal, 100); // atribui a sensVal o menor valor, seja sensVal ou 100

// garantindo que esse nunca seja maior que 100.

Notas e Advertências

Talvez contraintuitivamente, max() é constantemente usada para restringir o extremo inferior do intervalo de uma variável, enquanto min() é usado para restringir o extremo superior do intervalo.

Por causa da forma em que a função max() é implementada, evite usar outras funções dentro dos parênteses, isso pode levar a resultados incorretos.

min(a++, 100); // evitar isso - causa resultados incorretos

min(a, 100); // ao invés disso, usar esta forma

a++; // manter a aritmética fora da função

pow()

[Funções Matemáticas]

Descrição

Calcula o valor de um número elevado a uma potência. pow() pode ser usada para transformar um número em uma potência fracionária. Isso pode ser útil para gerar mapeamentos exponenciais de valores ou curvas.

Sintaxe

pow(base, expoente)

Parâmetros

base: o número (float)

expoente: a potência a qual o número deve ser elevado (float)

Retorna

O resultado da exponenciação (double)

Código de exemplo

Z recebe o valor de x elevado a y:

z = pow(x, y);

Veja o sketch (fscale) para um exemplo mais complexo do uso de pow().

sq()

[Funções Matemáticas]

Descrição

Calcula o quadrado de um número: o número multiplicado por si mesmo.

Sintaxe

sq(x)

Parâmetros

x: o número que se deseja calcular o quadrado, qualquer tipo de dados

Retorna

O quadrado do número. (double)

Notas e Advertências

Por causa da forma como a função sq() é implementada, evite usar outras funções dentro dos parênteses, pois isso pode levar a resultados incorretos.

Esse código gerará resultados incorretos:

int inputSquared = sq(Serial.parseInt()); // evite isso

Faça dessa forma:

int input = Serial.parseInt(); // manter outras operações fora da função sq()

int inputSquared = sq(input);

sqrt()

[Funções Matemáticas]

Descrição

Calcula a raiz quadrada de um número.

Sintaxe

sqrt(x)

Parâmetros

x: o número a se calcular a raiz quadrada, qualquer tipo de dados

Retorna

A raiz quadrada do número. (double)

Funções Trigonométricas

cos()

[Funções Trigonométricas]

Descrição

Calcula o cosseno de um ângulo (em radianos). O resultado é dado entre -1 e 1.

Sintaxe

cos(rad)

Parâmetros

rad: O ângulo em radianos (float).

Retorna

O cosseno do ângulo (double).

sin()

[Funções Trigonométricas]

Descrição

Calcula o seno de um ângulo (em radianos). O resultado é dado entre -1 e 1.

Sintaxe

sin(rad)

Parâmetros

rad: O ângulo em radianos (float).

Retorna

O seno do ângulo (double).

tan()

[Funções Trigonométricas]

Descrição

Calcula a tangente de um ângulo (em radianos). O resultado é dado entre -infinito e +infinito (limitado pela tamanho do tipo de dado double).

Sintaxe

tan(rad)

Parâmetros

rad: O ângulo em radianos (float).

Retorna

A tangente do ângulo (double).

Caracteres

isAlpha()

[Caracteres]

Descrição

Analisa se um caractere é alfabético (isto é, se é uma letra). Retorna true (verdadeiro) se thisChar contém uma letra.

Sintaxe

isAlpha(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um caractere alfabético.

Código de Exemplo

if (isAlpha(meuChar)) { // testa se a variável meuChar é uma letra

Serial.println("The character is a letter");

}

else {

Serial.println("The character is not a letter");

}

isAlphaNumeric()

[Caracteres]

Descrição

Analisa se um caractere é alfanumérico (isto é, uma letra ou um número). Retorna true (verdadeiro) se thisChar contém ou uma letra ou um número.

Sintaxe

isAlphaNumeric(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se a variável thisChar é uma letra ou um número..

Código de Exemplo

if (isAlphaNumeric(meuChar)) { // testa se meuChar é uma letra ou um número

Serial.println("The character is alphanumeric");

}

else {

Serial.println("The character is not alphanumeric");

}

isAscii()

[Caracteres]

Descrição

Analisa se um caractere é Ascii. Retorna true se a variável thisChar contém um caractere Ascii.

Sintaxe

isAscii(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um caractere Ascii.

Código de Exemplo

if (isAscii(meuChar)) { // testa se meuChar é um caractere Ascii

Serial.println("The character is Ascii");

}

else {

Serial.println("The character is not Ascii");

}

isControl()

[Caracteres]

Descrição

Analisa se uma caractere é um caractere de controle. Retorna true (verdadeiro) se thisChar é um caractere de controle.

Sintaxe

isControl(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um caractere de controle.

Código de Exemplo

if (isControl(meuChar)) { // testa se a variável meuChar é um caractere de controle

Serial.println("The character is a control character");

}

else {

Serial.println("The character is not a control character");

}

isDigit()

[Caracteres]

Descrição

Analisa se uma caractere é um digito (isto é, um número). Retorna true (verdadeiro) se thisChar é um número.

Sintaxe

isDigit(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um número.

Código de Exemplo

if (isDigit(meuChar)) { // testa se meuChar é um digito

Serial.println("The character is a number");

}

else {

Serial.println("The character is not a number");

}

isGraph()

[Caracteres]

Descrição

Analisa se um caractere é imprimível com algum conteúdo (espaços são imprimíveis, mas não possuem conteúdo). Retorna true se thisChar é imprimível.

Sintaxe

isGraph(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é imprimível.

Código de Exemplo

if (isGraph(meuChar)) { // testa se meuChar é um caractere imprimível mas não um espaço.

Serial.println("The character is printable");

}

else {

Serial.println("The character is not printable");

}

isHexadecimalDigit()

[Caracteres]

Descrição

Analisa se uma caractere é um digito hexadecimal (A-F, 0-9). Retorna true se thisChar contém um digito hexadecimal.

Sintaxe

isHexadecimalDigit(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um digito hexadecimal.

Código de Exemplo

if (isHexadecimalDigit(meuChar)) { // testa se meuChar é um digito hexadecimal

Serial.println("The character is an hexadecimal digit");

}

else {

Serial.println("The character is not an hexadecimal digit");

}

isLowerCase()

[Caracteres]

Descrição

Analisa se uma caractere é minúsculo (isto é, uma letra minúscula). Retorna true se thisChar contém uma letra minúscula.

Sintaxe

isLowerCase(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é minúsculo.

Código de Exemplo

if (isLowerCase(meuChar)) { // testa se meuChar é uma letra minúscula

Serial.println("The character is lower case");

}

else {

Serial.println("The character is not lower case");

}

isPrintable()

[Caracteres]

Descrição

Analisa se uma caractere é imprimível (isto é, qualquer caractere que produz uma saída, até mesmo um espaço). Retorna true se thisChar é imprimível. Por impimível diz-se os caracteres que pode ser impressos, como letras e números. Alguns caracteres são para controle e não podem ser impressos como o new line ('\n') e o tab ('\t').

Sintaxe

isPrintable(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é imprimível.

Código de Exemplo

if (isPrintable(meuChar)) { // testa se meuChar é imprimível

Serial.println("The character is printable");

}

else {

Serial.println("The character is not printable");

}

isPunct()

[Caracteres]

Descrição

Analisa se uma caractere é pontuação (isto é, uma vírgula, um ponto-e-vírgula, um ponto de exlamação etc). Retorna true se thisChar é um caractere de pontuação.

Sintaxe

isPunct(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um caractere de pontuação

Código de Exemplo

if (isPunct(meuChar)) { // testa se meuChar é um caractere de pontuação

Serial.println("The character is a punctuation");

}

else {

Serial.println("The character is not a punctuation");

}

isSpace()

[Caracteres]

Descrição

Analisa se uma caractere é o caractere de espaço. Retorna true se thisChar contém um espaço.

Sintaxe

isSpace(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é o caractere de espaço.

Código de Exemplo

if (isSpace(meuChar)) { // testa se meuChar é o caractere de espaço

Serial.println("The character is a space");

}

else {

Serial.println("The character is not a space");

}

isUpperCase()

[Caracteres]

Descrição

Analisa se uma caractere é maiúsculo (isto é, uma letra maisúcula). Retorna true se thisChar é uma letra maiúscula.

Sintaxe

isUpperCase(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é uma letra maiúscula.

Código de Exemplo

if (isUpperCase(meuChar)) { // testa se meuChar é uma letra maiúscula

Serial.println("The character is upper case");

}

else {

Serial.println("The character is not upper case");

}

isWhitespace()

[Caracteres]

Descrição

Analisa se um caractere é um espaço em branco, isto é o próprio caractere de espaço (' '), formfeed ('\f'), nova linha ('\n'), retorno ('\r'), tab horizontal tab ('\t') e tab vertical ('\v')). Retorna true se thisChar contém um espaço em branco.

Sintaxe

isWhitespace(thisChar)

Parâmetros

thisChar: variável. Tipos de dados permitidos: char

Retorna

true: se thisChar é um caractere de espaço em branco.

Código de Exemplo

if (isWhitespace(meuChar)) { // testa se meuChar é um caractere de espaço em branco

Serial.println("The character is a white space");

}

else {

Serial.println("The character is not a white space");

}

Números Aleatórios

random()

[Números Aleatórios]

Descrição

A função random() gera números pseudoaleatórios.

Sintaxe

random(max) random(min, max)

Parâmetros

min - menor limite do valor aleatório, inclusivo e opcional (long)

max - maior limite do valor aleatório, exclusivo (long)

Retorna

Um número inteiro aleatório entre min e max-1 (long) .

Código de exemplo

O código abaixo gera números aleatórios e os imprime na porta serial.

long randNumber;

void setup() {

Serial.begin(9600);

// Se o pino de entrada analógica 0 é deixado desconectado,

// o ruído aleatório analógico irá causar a chamada de randomSeed()

// gerar sementes aleatórias diferentes cada vez que o sketch roda.

// randomSeed() basicamente "embaralha" a função random().

randomSeed(analogRead(0));

}

void loop() {

// imprime um número aleatório entre 0 e 299

randNumber = random(300);

Serial.println(randNumber);

// imprime um valor aleatório entre 10 e 19

randNumber = random(10, 20);

Serial.println(randNumber);

delay(50);

}

randomSeed()

[Números Aleatórios]

Descrição

randomSeed() inicializa o gerador de números pseudoaleatórios, fazendo o começar em um ponto arbitrário em sua sequência aleatória. Essa sequência, enquanto muito longa, e aleatória, é sempre a mesma.

Se é importante que uma sequência de valores gerados por random() seja diferente em execuções subsequentes de um sketch, use randomSeed() para inicializar o gerador de números aleatórios com uma entrada significantemente aleatória, como analogRead() em um pino desconectado.

Por outro lado, pode ser ocasionalmente útil usar sequências pseudoaleatórias exatamente repetidas. Isso pode ser conseguido chamando-se randomSeed() com um número fixo, antes de começar a usar a sequência aleatória.

Parâmetros

seed - número para inicializar a sequência pseudoaleatória (long).

Retorna

Nada

Código de Exemplo

O código abaixo imprime números aleatórios na porta serial.

long randNumber;

void setup() {

// inicializa a porta serial.

Serial.begin(9600);

// inicializa o gerador de números aleatórios.

// um pino analógico desconectado irá retornar um

// valor aleatório de tensão em analogRead()

randomSeed(analogRead(0));

}

void loop() {

// calcula o próximo número aleatório entre 0 e 299

randNumber = random(300);

// imprime o valor na porta serial

Serial.println(randNumber);

delay(50);

}

Bits e Bytes

bit()

[Bits e Bytes]

Descrição

Computa o valor do bit especificado (o bit 0 é igual a 1, bit 1 igual a 2, bit 2 igual a 4, etc.).

Sintaxe

bit(n)

Parâmetros

n: o bit o qual se deseja computar o valor

Retorna

O valor do bit.

bitClear()

[Bits e Bytes]

Descrição

Limpa (escreve um 0) em um bit de uma variável numérica.

Sintaxe

bitClear(x, n)

Parâmetros

x: a variável numérica da qual se deseja limpar um bit

n: qual bit limpar, começando em 0 para o bit menos significativo (o bit mais à direta)

Retorna

Nada

bitRead()

[Bits e Bytes]

Descrição

Lê o valor de um bit em um número.

Sintaxe

bitRead(x, n)

Parâmetros

x: o número do qual se quer ler o bit

n: qual bit ler, começando em 0 para o bit menos significativo (mais à direita)

Retorna

O valor do bit (0 ou 1).

bitSet()

[Bits e Bytes]

Ativa (escreve 1 em) um bit de uma variável numérica.

Descrição

Sintaxe

bitSet(x, n)

Parâmetros

x: a variável numérica cujo bit se quer ativar

n: qual bit ativar, começando em 0 para o bit menos significativo (mais à direita)

Retorna

Nada

bitWrite()

[Bits e Bytes]

Descrição

Escreve em um bit especificado de um valor numérico.

Sintaxe

bitWrite(x, n, b)

Parâmetros

x: a variável numérica cujo bit se deseja escrever

n: qual bit do número a ser escrito, começando em 0 para o bit menos significativo (mais à direita)

b: o valor a ser escrito no bit (0 ou 1)

Retorna

Nada

Código de Exemplo

O código a seguir demonstra o use de bitWrite ao imprimir uma variável no Monitor Serial antes e depois do uso de bitWrite().

void setup() {

Serial.begin(9600);

while (!Serial) {} // espera a porta serial conectar. Necessário apenas em placas com USB nativa

byte x = 0b10000000; // o prefixo 0b indica uma constante binária

Serial.println(x, BIN); // 10000000

bitWrite(x, 0, 1); // Escreve 1 no biit menos significante de x

Serial.println(x, BIN); // 10000001

}

void loop() {}

highByte()

[Bits e Bytes]

Descrição

Extrai o byte mais significativo (o mais à esquerda) de uma word (valor de 16 bits), ou o segundo byte menos significativo de um tipo de dado maior.

Sintaxe

highByte(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

byte

lowByte()

[Bits e Bytes]

Descrição

Extrai o byte menos significativo (mais à direita) de uma variável (ex. uma word).

Sintaxe

lowByte(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

byte

Interrupções Externas

attachInterrupt()

[Interrupções Externas]

Descrição

Pinos Digitais com Interrupções

O primeiro parâmetro de attachInterrupt() é o número da interrupção. É recomendado usar digitalPinToInterrupt(pino) para converter o número do pino digital para o número específico da interrupção. Por exemplo, se você usar o pino 3, passe digitalPinToInterrupt(3) como o primeiro parâmetro de attachInterrupt().

PLACA PINOS DIGITAIS POSSÍVEIS PARA INTERRUPÇÕES

Uno, Nano, Mini e outros com 328p

2, 3

Uno WiFi Rev.2

todos os pinos digitais

Mega, Mega2560, MegaADK

2, 3, 18, 19, 20, 21

Micro, Leonardo e outros com 32u4

0, 1, 2, 3, 7

Zero

todos os pinos digitais, exceto 4

Família MKR

0, 1, 4, 5, 6, 7, 8, 9, A1, A2

Due

todos os pinos digitais

101

todos os pinos digitais (Apenas pinos 2, 5, 7, 8, 10, 11, 12, 13 funcionam com CHANGE)

Para as placas Uno WiFiRev.2, Due, Zero, Família MKR e 101, número da interrupção = número do pino.

Notas e Advertências

Nota

Dentro da função associada, delay() não funcionará e o valor retornado por millis() não será incrementado. Dados recebidos na porta serial enquanto a interrupção é executada podem ser perdidos. Você deve declarar como volatile quaisquer que você modificar dentro da função associada. Veja a seção sobre ISRs abaixo para mais informação.

Usando Interrupções

Interrupções são úteis para fazer coisas automaticamente em programas de microcontroladores, e podem ajudar a resolver problemas de temporização. Boas tarefas para se udar uma interrupção podem icnluir a leitura de um codificador rotativo, ou monitorar entradas de dados pelo usuário.

Supomos que você quisesse ter certeza que um programa sempre captura os pulsos de um codifcador rotativo, sem nunca perder um pulso, seria muito complicado criar um programa que pudesse fazer qualquer outra coisa além disso, porque o programa precisaria checar constantemente os pinos do codificador, para capturar os pulsos exatamente quando eles ocorreram. Outros sensores são similarmente dinâmicos também, como um sensor de som que pode ser usado para capturar um clique, ou um sensor infravermelho (foto-interruptor) para capturar a queda de uma moeda. Em todas essas situações, usar uma interrupção pode permitir o microcontrolador trabalhar em outras tarefas sem perder a interrupção.

Sobre Rotinas de Serviço de Interrupções (ISRs)

ISRs são tipos especiais de funções que possuem algumas limitações únicas que as outras funções não possuem. Uma ISR não recebe argumentos, e não devem retornar nada.

Geralmente, uma ISR deve ser o mais curta e rápida possível. Se o seu sketch usa múltiplas ISRs, apenas uma pode ser executada de cada vez, outras interrupções serão executadas após a atual terminar, em uma ordem que depende de sua prioridade. millis() depende de interrupções para contar, então essa nunca irá incrementar dentro de uma ISR. Como delay() também depende de interrupções, não irá funcionar dentro de uma ISR. micros() funciona inicialmente, mas começará a se comportar erraticamente após 1-2 ms. delayMicroseconds() não usa nenhum contador, então funciona normalmente.

Tipicamente variáveis globais são usadas para passar dados entre uma ISR e o programa principal. Tenha certeza que variáveis compartilhadas entre uma ISR e o programa principal são atualizadas corretamente, para isso, as declare usando o modificador volatile.

Para mais informações sobre interrupções, veja as notas de Nick Gammon’s (Em Inglês).

Sintaxe

attachInterrupt(digitalPinToInterrupt(pino), ISR, modo); (Recomendado)

attachInterrupt(interrupção, ISR, modo); (Não recomendado)

attachInterrupt(pino, ISR, modo); (Não recomendado. Além disso, essa sintaxe só funciona em placas Arduino SAMD, Uno WiFi Rev2, Due, e 101)

Parâmetros

interrupção: o número da interrupção (int)

pino: o número do pino

ISR: a ISR a ser chamada quando a interrupção ocorre; essa função deve não tomar nenhum parâmetro nem retornar nada. Essa função é chamada de rotina de serviço da interrupção ou ISR (do Inglês, interrupt service routine).

modo: define quando a interrupção deve ser ativada. Quatro constantes estão predefinidas como valores válidos:

LOW acionar a interrupção quando o estado do pino for LOW,

CHANGE acionar a interrupção quando o sempre estado do pino mudar

RISING acionar a interrupção quando o estado do pino for de LOW para HIGH apenas,

FALLING acionar a interrupção quando o estado do pino for de HIGH para LOW apenas.

Placas Due, Zero e MKR1000 suportam também:

HIGH acionar a interrupção quando o estado do pino for HIGH.

Retorna

Nada

Código de Exemplo

O código abaixo usa uma interrupção para capturar a mudança no estado do pino 2 e acender o LED de acordo.

const byte ledPin = 13;

const byte interruptPin = 2;

volatile byte state = LOW;

void setup() {

pinMode(ledPin, OUTPUT);

pinMode(interruptPin, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);

}

void loop() {

digitalWrite(ledPin, state);

}

void blink() {

state = !state;

}

Números das Interrupções

Normalmente você deve usar digitalPinToInterrupt(pino), em vez de colocar diretamente o número da interrupção no seu sketch. Os pinos específicos para interrupções, e seu mapeamento para o número da interrupção variam para cada tipo de placa. O uso direto dos números das interrupções pode parecer mais simples, porém pode causar problemas de compatibilidade quando seu sketch for executado em uma placa diferente.

Mesmo assim, sketches mais antigos frequentemente têm os números das interrupções usados diretamente. Frequentemente o número 0 (para o pino digital 2) ou o número 1 (para o pino digital 3) foram usados. A tabela abaixo mostra os pinos com interrupções disponíveis em cada placa.

Note que na tabela abaixo, os números das interrupções se referem aos números a serem passados para attachInterrupt(). Por razões históricas, essa numeração nem sempre corresponde diretamente a numeração do chip ATmega (ex. int.0 corresponde à INT4 no chip ATmega2560).

PLACA INT.0 INT.1 INT.2 INT.3 INT.4 INT.5

Uno, Ethernet

2

3

Mega2560

2

3

21

20

19

18

Leonardo, Micro (32u4)

3

2

0

1

7

Para as placas Due, Zero, MKR1000 e 101 o número da interrupção = número do pino.

detachInterrupt()

[Interrupções Externas]

Descrição

Desativa a interrupção especificada.

Sintaxe

detachInterrupt(digitalPinToInterrupt(pino)) (Recomendado)

detachInterrupt(interrupt) (Não recomendado)

detachInterrupt(pino) (Não recomendado. Além disso, essa sintaxe só funciona em placas Arduino SAMD, Uno WiFi Rev2, Due, e 101)

Parâmetros

interrupt: o número da interrupção a ser desativada (veja attachInterrupt() para mais detalhes).

pino: o número do pino da interrupção a ser desativada

Retorna

Nada

Interrupções

interrupts()

[Interrupções]

Descrição

Reativa interrupções (quando elas tiverem sido desativadas por nointerrupts(). Interrupções permitem certas tarefas importantes acontecerem ao fundo e são, por padrão, ativadas. Algumas funções não irão funcionar enquanto as interrupções estiverem desativadas, e dados recebidos podem ser ignorados. Interrupções podem levemente interferir no timing do código, no entanto, e podem ser desativadas em seções particulamente críticas do código.

Sintaxe

interrupts()

Parâmetros

Nothing

Retorna

Nada

Código de Exemplo

O código abaixo mostra como desativar e reativar interrupções.

void setup() {}

void loop() {

noInterrupts();

// código crítico e sensível ao tempo aqui

interrupts();

// códigos regulares aqui

}

noInterrupts()

[Interrupções]

Descrição

Desativa interrupções (você pode reativá-las com interrupts()). Interrupções permitem certas tarefas importantes acontecerem ao fundo e são, por padrão, ativadas. Algumas funções não irão funcionar enquanto as interrupções estiverem desativadas, e dados recebidos podem ser ignorados. Interrupções podem levemente interferir no timing do código, no entanto, e podem ser desativadas em seções particulamente críticas do código.

Sintaxe

noInterrupts()

Parâmetros

Nothing

Retorna

Nada

Código de Exemplo

O código abaixo mostra como desativar e reativar interrupções.

void setup() {}

void loop() {

noInterrupts();

// código crítico e sensível ao tempo aqui

interrupts();

// códigos regulares aqui

}

Comunicação

Serial

[Comunicação]

Description

Usada para comunicação entre uma placa Arduino e um computador ou outros dispositivos. Todas as placas Arduino possuem pelo menos uma porta serial (também conhecida como UART ou USART), enquanto alguns possuem várias .

PLACA NOME USB CDC PINOS - SERIAL PINOS - SERIAL1 PINOS - SERIAL2 PINOS - SERIAL3

Uno, Nano, Mini

0(RX), 1(TX)

Mega

0(RX), 1(TX)

19(RX), 18(TX)

17(RX), 16(TX)

15(RX), 14(TX)

Leonardo, Micro, Yún

Serial

0(RX), 1(TX)

Uno WiFi Rev.2

Conectados a USB

0(RX), 1(TX)

Conectados a NINA

Família MKR

Serial

13(RX), 14(TX)

Zero

SerialUSB (USB nativa apenas)

Conect. a porta de gravação

0(RX), 1(TX)

Due

SerialUSB (USB nativa apenas)

0(RX), 1(TX)

19(RX), 18(TX)

17(RX), 16(TX)

15(RX), 14(TX)

101

Serial

0(RX), 1(TX)

No Uno, Nano, Mini, e Mega, pinos 0 e 1 são usados para comunicação com o computador. Conectar qualquer coisa a esses pinos pode interferir nessa comunicação, incluindo causar falhas na gravação da placa.

Você pode usar o monitor serial integrado na IDE do Arduino para comunicar-se com uma placa Arduino. Clique no botão do monitor serial (lupa) na barra de ferramentas e escolha a mesma taxa de tranmissão usada na chamada de begin().

Você pode usar o monitor serial presente na IDE do Arduino para se comunicar com uma placa Arduino. Clique no botão do monitor serial na barra de ferramentas e escolha e usar a mesma taxa de transferência (baud rate) usada na chamada de begin().

Comunicação serial nos pinos TX/RX usa níveis lógicos TTL (5V ou 3.3V dependendo da placa). Não conecte esses pinos diretamente um uma porta serial RS232; esses operam em +/- 12V e podem danificar sua placa Arduino.

Para usar esses pinos para comunicar com seu computador pessoal, você vai precisar de um adaptador USB-serial extra, pois esses pinos não são conectados ao adaptador USB-serial do Arduino Mega. para usá-los para comunicar com um dispositivo serial TTL externo, conecte o pino TX do Arduino ao pino RX do dispositivo, o pino RX ao pino TX do dispositivo, e o ground de seu Mega ao ground do dispositivo.

Funções

if(Serial)

available()

availableForWrite()

begin()

end()

find()

findUntil()

flush()

parseFloat()

parseInt()

peek()

print()

println()

read()

readBytes()

readBytesUntil()

readString()

readStringUntil()

setTimeout()

write()

serialEvent()

Stream

[Comunicação]

Descrição

Stream é a classe base para streams de caracteres ou binárias. Não é chamada diretamente, mas invocada quando você usa uma função que depende dessa classe.

Stream define as funções de leitura no Arduino. Quando usar qulquer funcionalidade core que usa read() ou um método similar, você pode assumir seguramente que essa faz chamadas a classe Stream. Para funções como print(), a classe Stream herda da classe Print.

Algumas das bibliotecas que dependem da classe Stream são:

Serial

Wire

Ethernet

SD

Funções

available()

read()

flush()

find()

findUntil()

peek()

readBytes()

readBytesUntil()

readString()

readStringUntil()

parseInt()

parseFloat()

setTimeout()

USB

Keyboard

[USB]

Descrição

As funções da biblioteca Keyboard permitem placas baseadas nos micros 32u4 ou SAMD funcionar como um teclado e enviar sinais de teclas pressionadas a um computador conectado através da porta micro USB nativa.

Nota: Nem todo caractere ASCII, particularmente os não imprimíveis, podem ser enviados com a biblioteca Keyboard.

A biblioteca suporta o uso de teclas modificadoras. Teclas modificadoras mudam o comportamento de outra tecla quando pressionadas simultaneamente. Veja aqui informações adicionais sobre as teclas suportadas e seu uso.

Notas e Advertências

As bibliotecas core permitem as placas baseadas nos micros 32u4 e SAMD (Leonardo, Esplora, Zero, Due e família MKR) aparecer em um computador conectado como um mouse e/ou teclado nativo.

Uma nota as bibliotecas Mouse e Keyboard: se a biblioteca Mouse ou Keyboard estiver rodando constantemente, será difícil programar a sua placa. Funções como Mouse.move() e Keyboard.print() irão mover o cursor ou enviar teclas ao computador conectado e devem ser chamadas apenas quando você estiver pronto para usá-las. É recomendado se usar um sistema de controle para ativar a funcionalidade USB, como um botão físico ou apenas responder a entradas específicas que você pode controlar.Confira os exemplos sobre as bibliotecas Mouse e Keyboard para algumas formas de fazer isso.

Quando usar as bibliotecas Mouse ou Keyboard, pode ser melhor testar sua saída primeiro utilizando Serial.print(). Dessa forma, você pode ter certeza de quais valores estão sendo relatados.

Funções

Keyboard.begin()

Keyboard.end()

Keyboard.press()

Keyboard.print()

Keyboard.println()

Keyboard.release()

Keyboard.releaseAll()

Keyboard.write()

Mouse

[USB]

Descrição

As funções da biblioteca mouse permitem placas baseadas nos micros 32u4 ou SAMD controlar o movimento do cursor em um computador conectado através da porta micro USB nativa. Quando a posição do cursor é atualizada, essa é sempre relativa a sua posição anterior.

Notas e Advertências

As bibliotecas core permitem as placas baseadas nos micros 32u4 e SAMD (Leonardo, Esplora, Zero, Due e família MKR) aparecer em um computador conectado como um mouse e/ou teclado nativo.

Uma nota as bibliotecas Mouse e Keyboard: se a biblioteca Mouse ou Keyboard estiver rodando constantemente, será difícil programar a sua placa. Funções como Mouse.move() e Keyboard.print() irão mover o cursor ou enviar teclas ao computador conectado e devem ser chamadas apenas quando você estiver pronto para usá-las. É recomendado se usar um sistema de controle para ativar a funcionalidade USB, como um botão físico ou apenas responder a entradas específicas que você pode controlar. Confira os exemplos sobre as bibliotecas Mouse e Keyboard para algumas formas de fazer isso.

Quando usar as bibliotecas Mouse ou Keyboard, pode ser melhor testar sua saída primeiro utilizando Serial.print(). Dessa forma, você pode ter certeza de quais valores estão sendo relatados.

Funções

Mouse.begin()

Mouse.click()

Mouse.end()

Mouse.move()

Mouse.press()

Mouse.release()

Mouse.isPressed()

VARIÁVEIS

Tipos de dados e constantes da linguagem Arduino.

Constantes

Constantes

[Constantes]

Descrição

Constantes são expressões predefinidas na linguagem Arduino. Essas são usadas para tornar os programas mais legíveis. As constantes da linguagem Arduino são classificadas nos grupos:

Definições de Constantes Booleanas: true e false

Existem duas constantes usadas para representar verdade e falsidade na linguagem Arduino: true e false.

false

false é o mais fácil dos dois de ser definido. Sendo definido apenas como 0 (zero).

true

true é freqeuentemente dito ser definido como 1, o que está correto, porém true possui uma definição mais ampla. Qualquer inteiro que não seja zero é true, em um sentido booleano. Então -1, 2 e -200 são todos definidos como true, também, em um sentido booleano.

Note que as constantes true e false são digitadas em caixa baixa, diferentemente de HIGH, LOW, INPUT, and OUTPUT.

Definições de níveis lógicos: HIGH e LOW

Quando lendo-se ou escrevendo-se o estado de um pino digital há apenas dois valores possíveis que esse pino pode assumir: HIGH e LOW.

HIGH

O significado de HIGH (em relação a um pino) depende se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Qaundo um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar HIGH se:

uma tensão maior que 3.0V está presente no pino (em placas 5V)

uma tensão maior que 2.0V está presente no pino (em placas 3.3V)

Um pino pode também ser configurado como entrada (INPUT) com pinMode(), e posteriormente colocado em HIGH com digitalWrite(). Isso irá ativar os resistores de 20K internos para pullup, o que irá forçar o estado do pino para HIGH a menos que esse seja forçado para LOW por um circuito externo. Isso é exatamente como a opção INPUT\_PULLUP funciona, sendo descrita abaixo em mais detalhes.

Quando um pino é configurado como saída (OUTPUT) com pinMode(), e colocado em estado HIGH com digitalWrite(), a tensão no pino é:

5 volts (em placas 5V)

3.3 volts (em placas 3.3V)

Nesse estado, o pino pode fornecer corrente (ver limites para sua placa, geralmente 40.0mA). Isso pode ser usado, por exemplo, para acender um LED que está conecatdo ao ground através de um resistor limitador de corrente.

LOW

O significado de LOW depende também se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Qaundo um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar LOW se:

uma tensão menor que 1.5V está presente no pino (em placas 5V)

uma tensão menor que 1.0V (aproxidamente) está presente no pino (em placas 3.3V)

Quando um pino é configurado como OUTPUT com pinMode(), e colocado em estado LOW com digitalWrite(), a tensão no pino é 0 volts (tanto em placas de 5V como 3.3V boards). Nesse estado o pino pode absorver corrente, o que pode ser usado, por exemplo, para acender um LED conectado através de um resistor limitador de corrente aos +5 volts (ou +3.3 volts).

Definições de modos para Pinos Digitais: INPUT, INPUT\_PULLUP e OUTPUT

Pinos digitais pode ser usados como entrada, entrada com pull-up ou saída (INPUT, INPUT\_PULLUP ou OUTPUT, respectivamente). Mudar o modo de um pino com pinMode() muda o comportamento elétrico do pino.

Pinos Configurados como entrada - INPUT

Pinos configurados como INPUT com pinMode() são ditos estarem em um estado de alta-impedância. Quando configurados como INPUT, esses pinos exigem multíssimo pouco dos circuitos que estão verificando, equivalente a um resistor de 100 Megohm na frente do pino. Isso faz com que sejam muito úteis para ler sensores.

Se você configurar um pino como INPUT, e ler o estado de um botão conectado a 5V, por exemplo, quando o botão estiver em estado aberto, a entrada estará "flutuando", resultando em leituras imprevisíveis. Isso acontece porque quando o botão está aberto, o pino não está conectado a nenhuma tensão. Para assegurar uma leitura correta quando o botão está aberto, um resistor de pull-up ou pull-down deve ser usado. O propósito desse resistor é colocar o pino em um estado conhecido quando o botão estiver aberto. Um resistor de 10K ohm é usado geralmente, pois possui valor baixo o suficiente para previnir confiavelmente o estado flutuante, ao mesmo tempo que é grande o suficiente para não usar muita corrente qaundo o botão estiver fechado. Veja o tutorial Serial e Digital Read (Em Innglẽs) para mais informações.

Se um resistor de pull-down é usado, o pino de entrada estará em LOW quando o botão estiver aberto e HIGH quando o btoão estiver pressionado.

Se um resistor de pull-up é usado, o pino de entrada estará em HIGH quando o botão estiver aberto e LOW quando o btoão estiver pressionado.

Pinos Configuradoos como entrada com pull-up - INPUT\_PULLUP

O microcontrolador ATmega na maioria das placas Arduino possui resistores pull-up internos (resistores conectados a alimentação internamente) que você pode acessar. Se você preferir usar estes em vez de resistores de pull-up externos, você pode usar o argumento INPUT\_PULLUP na função pinMode().

Veja o tutorial Input Pullup Serial (Em Inglês) para um exemplo do uso desses resistores.

Pinos configurados como entradas com ambos INPUT ou INPUT\_PULLUP podem ser danificados ou destruídos se são conectados a tensões abaixo de zero (tensões negativas) ou acima da tensão de alimentação (5V ou 3V).

Pinos Configurados como saída - OUTPUT

Pinos configurados como OUTPUT com pinMode() são ditos estarem em um estado de baixa-impedância. Isso significa que esses podem fornecer uma quantidade substancial de corrente para outros circuitos. Os pinos de um ATmega podem fornecer ou absorver correntes de até 40 mA (miliamperes) para/de outros dispositivos/circuitos. Isso faz com que sejam úteis para alimentar LEDs, pois LEDs tipicamente usam menos de 40 mA. Cargas que exigem mais de 40 mA (ex. motores) irão reqeurer um transistor ou um outro circuito de interface.

Pinos configurados como saídas podem ser danificados ou destruídos se forem conectados diretamente ao ground ou na tensão de alimentação.

Definição do LED na placa: LED\_BUILTIN

A maioria das placas Arduino possuem um pino conectado a um LED on-board através de um resistor. A constante LED\_BUILTIN é o número do pino ao qual o LED on-board está conectado. Na maioria das placas, esse LED é conectado ao pino digital 13.

Constantes Inteiras

[Constantes]

Descrição

Constantes Inteiras são números usados diretamente em um sketch, como 123. Por padrão, esses números são tratados como int mas você pode mudar isso com os modificadores U e L (ver abaixo).

Normalmente, constantes inteiras são tratadas como inteiros na base 10 (decimais), mas notações especiais (modificadores) podem ser usados para escrever-se números em outras bases.

BASE EXEMPLO MODIFICADOR COMENTÁRIO

10 (decimal)

123

nenhum

2 (binário)

B1111011

prefixo 'B'

funciona apenas com valores 8-bit (0 a 255) caracteres 0 e 1 válidos

8 (octal)

0173

prefixo "0"

caracteres 0-7 válidos

16 (hexadecimal)

0x7B

prefixo "0x"

caracteres 0-9, A-F, a-f válidos

Decimal (base 10)

Essa é a matemática de senso comum a qual você já está acostumado. Constantes sem prefixos são assumidas como valores decimais.

Código de Exemplo:

n = 101; // o mesmo que 101 decimal ((1 \* 10^2) + (0 \* 10^1) + 1)

Binário (base 2)

Apenas os caracteres 0 e 1 são válidos.

Código de Exemplo:

n = B101; // o mesmo que 5 decimal ((1 \* 2^2) + (0 \* 2^1) + 1)

O modificador binário funciona apenas em bytes (8 bits), ou seja, entre 0 (B0) e 255 (B11111111). Se for conveniente inserir um int (16 bits) em formato binário, você pode fazê-lo com um procedimento em dois passos, como:

meuInt = (B11001100 \* 256) + B10101010; // B11001100 é o byte mais significativo

Octal (base 8)

Apenas os caracteres 0 a 7 são válidos. Valores octais são indicados pelo prefixo "0" (zero).

Código de Exemplo:

n = 0101; // o mesmo que 65 decimal ((1 \* 8^2) + (0 \* 8^1) + 1)

Cuidado: é possível criar um bug difícil de encontrar (acidentalmente), caso seja inserido um zero antes de uma constante, fazendo com que o compilador interprete sua constante como octal.

Hexadecimal (base 16)

Os caracteres válidos são 0 a 9 e as letras A a F; A possui valor 10, B é 11, até o F, que é 15. Valores hexadecimais são indicados pelo prefixo "0x". Note que as letras A-F podem ser também iseridas em minúsculo (a-f).

Código de Exemplo:

n = 0x101; // o qmesmo que 257 decimal ((1 \* 16^2) + (0 \* 16^1) + 1)

Notas e Advertências

Modificadores 'U' e 'L':

Por padrão, uma constante inteira é tratada como int com as mesmas limitações desse tipo de dado. Para especificar uma constante inteira com outro tipo de dado, termine-a com:

um 'u' ou 'U' para forçar a constante para formato unsigned (sem sinal). Exemplo: 33u

um 'l' ou 'L' para forçar a constante para formato long. Exemplo: 100000L

um 'ul' ou 'UL' para forçar a constante para formato unsigned long. Exemplo: 32767ul

Constantes de Ponto Flutuante

[Constantes]

Descrição

Similar as constantes inteiras, constantes de ponto flutuante são usadas para tornar o código mais legível. Constantes de ponto flutuante são trocadas em tempo de compilação para o valor calculado para a expressão.

Código de Exemplo

n = 0.005; // 0.005 é uma constante de ponto flutuante

Notas e Advertências

Constantes de ponto flutuante podem ser também expressas em uma variedade de notações científicas. 'E' e 'e' são ambas aceitas como indicadores de expoente.

CONSTANTE DE PONTO FLUTUANTE INTERPRETADA COMO: INTERPRETADA TAMBÉM COMO:

10.0

10

2.34E5

2.34 \* 10^5

234000

67e-12

67.0 \* 10^-12

0.000000000067

Conversão

byte()

[Conversão]

Descrição

Converte um valor para o tipo de dado byte.

Sintaxe

byte(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

o valor passado para a função, convertido para o tipo byte

char()

[Conversão]

Descrição

Converte um valor para o tipo de dado char.

Sintaxe

char(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

o valor passado para a função, convertido para o tipo char

float()

[Conversão]

Descrição

Converte um valor para o tipo de dado float.

Sintaxe

float(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

o valor passado para a função, convertido para o tipo float

Notas e Advertências

Veja a página da referência para o tipo float para detalhes sobre a precisão e limitações dos números de ponto flutuante no Arduino.

int()

[Conversão]

Descrição

Converte um valor para o tipo de dado int.

Sintaxe

int(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

O valor passado para a função, convertido para o tipo int

long()

[Conversão]

Descrição

Converte um valor para o tipo de dado long.

Sintaxe

long(x)

Parâmetros

x: um valor de qualquer tipo

Retorna

O valor passado para a função, convertido para o tipo long

word()

[Conversão]

Descrição

Converte um valor para o tipo de dado word.

Sintaxe

word(x)

word(h, l)

Parâmetros

x: um valor de qualquer tipo

h: o byte mais significativo (mais à esquerda) da palavra

l: o byte menos significativo (mais à direita) da palavra

Retorna

O valor passado para a função, convertido para o tipo word

Tipos de Dados

String()

[Tipos de Dados]

Descrição

Constrói uma instância da classe String. Há múltiplas versões que constroem Strings a partir de diferentes tipos de dados (ex. formatam-nos como uma sequência de caracteres), incluindo:

uma string constante de caracteres, em aspas duplas (ex. um vetor de caracteres)

um único caractere constante, em aspas simples

uma outra instância de um objeto String

um int constante ou long int

um int constante ou long int, usando uma base especificada

uma variável int ou long int

uma variável int ou long int, usando uma base especificada

um float ou double, usando um número especificado de casas decimais

Nesse texto, sequências de caracteres são chamadas strings, enquanto instâncias da classe String (objetos) são chamados String.

Construir uma String a partir de um número resulta em uma String que contém a representação ASCII desse número. A base padrão é dez, então:

String thisString = String(13);

Fornece a String "13". No entanto, você pode usar outras bases. Por exemplo,

String thisString = String(13, HEX);

que fornece a String "D", que é a representação hexadecimal do valor 13. Ou se você preferir binário,

String thisString = String(13, BIN);

forneca a String "1101", que é a representação binária de 13.

Sintaxe

String(val)

String(val, base)

String(val, decimalPlaces)

Parâmetros

val: a variável a ser formatada como String - Tipos de dados permitidos: string, char, byte, int, long, unsigned int, unsigned long, float, double

base (opcional): a base na qual formatar valores inteiros decimalPlaces (apenas se val é float ou double): o número desejado de casas decimais

Retorna

Uma instância da classe String.

Código de Exemplo

Todas a seguir são declarações válidas para Strings.

String stringOne = "Hello String"; // usando uma string constante

String stringOne = String('a'); // convertendo um caractere constante para uma String

String stringTwo = String("This is a string"); // convertendo uma string constante para um objeto String

String stringOne = String(stringTwo + " with more"); // concatenando duas strings

String stringOne = String(13); // usando um inteiro constante

String stringOne = String(analogRead(0), DEC); // usando um int e uma base especificada (decimal)

String stringOne = String(45, HEX); // usando um int e uma base especificada (hexadecimal)

String stringOne = String(255, BIN); // usando um int e uma base especificada (binário)

String stringOne = String(millis(), DEC); // usando um long e uma base especificada

String stringOne = String(5.698, 3); // usando um float e o número de casas decimais

bool

[Tipos de Dados]

Descrição

O tipo bool pode armazenar dois valores: true or false. (Cada variável bool ocupa um byte na memória.)

Sintaxe

bool var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O código abaixo mostra como usar o tipo de dado bool.

int LEDpin = 5; // LED no pino digital 5

int switchPin = 13; // botão conectado ao pino 13 e o outro terminal ao ground

bool running = false; //variável de tipo booleano

void setup() {

pinMode(LEDpin, OUTPUT);

pinMode(switchPin, INPUT);

digitalWrite(switchPin, HIGH); // ativa o resistor pull-up

}

void loop() {

if (digitalRead(switchPin) == LOW) {

// botão foi pressionado, o pull-up mantém o pino em HIGH internamente

delay(100); // delay para fazer o debounce no botão

running = !running; // troca o valor da variável running

digitalWrite(LEDpin, running); // indica via LED

}

}

boolean

[Tipos de Dados]

Descrição

boolean é um alias de tipos de dados não padrão para bool definido pelo Arduino. É recomendado usar em vez disso o tipo padrão bool, que é idêntico.

Código de Exemplo

O código abaixo mostra como usar o tipo de dado boolean.

int LEDpin = 5; // LED no pino digital 5

int switchPin = 13; // botão conectado ao pino 13 e o outro terminal ao ground

boolean running = false; //variável do tipo boolean

void setup()

{

pinMode(LEDpin, OUTPUT);

pinMode(switchPin, INPUT);

digitalWrite(switchPin, HIGH); // ativa o resistor pull-up

}

void loop()

{

if (digitalRead(switchPin) == LOW)

{ // botão foi pressionado, o pull-up mantém o pino em HIGH internamente

delay(100); // delay para fazer o debounce no botão

running = !running; // troca o valor da variável running

digitalWrite(LEDpin, running); // indica via LED

}

}

byte

[Tipos de Dados]

Descrição

Uma variável 'byte' armazena valores numéricos de 8-bit sem sinal, de 0 a 255.

Sintaxe

byte var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

char

[Tipos de Dados]

Descrição

Um tipo de dado usado para armazenar um caractere. Caracteres literais são escritos em aspas simples, dessa forma: 'A' (para múltiplos caracteres - ou seja, strings - use aspas duplas: "ABC").

No entanto, caracteres são armazenados como números. Você pode ver a codificação na tabela ASCII. Isso significa que é possível fazer aritmética com caracteres, na qual o valor ASCII para o caractere é usado (ex. 'A' + 1 tem valor 66, pois o valor ASCII da letra maiúscula A é 65). Veja a referência de Serial.println para mais sobre como caracteres são traduzidos para números.

O tipo de dado char ocupa ao menos 8 bits. É recomendado que se utilize char apenas para armazenar caracteres. Para um tipo de dado sem sinal, de um byte (8 bits), use o tipo de dado byte.

Sintaxe

char var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

char myChar = 'A';

char myChar = 65; // ambas as formas são equivalentes

double

[Tipos de Dados]

Descrição

Número de Ponto flutuante Double. No UNO e outras placas baseadas no ATMEGA, ocupa 4 bytes. Isto é, nesses a impelmentação do double é exatamente a mesma do float, sem nenhum ganho em precisão.

No Arduino Due e outras placas com SAMD, doubles usam 8 bytes, ou seja, precisão de 64 bits.

Sintaxe

double var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Notas e Advertências

Usuários que pegam código de outras fontes que inclua variáveis double pidem desejar examinar o código para conferir se a precisão é diferente do que realmente é alcançado nas placas Arduino baseadas em microcontroladores ATMEGA.

float

[Tipos de Dados]

Descrição

Tipo de dado para números de ponto flutuante, ou seja, um número racional. Números de ponto flutuante são frequentemente usados para aproximar valores analógicos e contínuos porque possuem maior resolução que inteiros (int). Números de ponto flutuante podem ser tão grandes quanto 3.4028235E+38 ou pequenos quanto -3.4028235E+38. Floats são armazenados em 32 bits (4 bytes) de memória.

Floats possuem apenas 6-7 dígitos decimais de precisão. Isso para o número total de dígitos, não de dígitos à direita do ponto decimal. Diferentemente de outras plataformas, onde se pode ter mais precisão usando-se um double (até 15 dígitos), no Arduino, o double possui o mesmo tamanho que float.

Números de ponto flutuante não são exatos, e podem gerar resultados estranhos quando comparados. Por exemplo, 6.0 / 3.0 pode não ser igual a 2.0. Você deveria, em vez disso, checar se o valor absoluto da diferença entre dois números é menor que algum número.

A matemática em números de ponto flutuante também é muito mais lenta que em inteiros ao se realizar cálculos, então deve ser evitada se, por exemplo, um loop tem que ser executado em alta velocidade para uma função de tempo crítico. Programadores frequentemente fazem o possível para converter cálculos em números flutuantes para matemática em inteiros para aumentar a velocidade, mas sacrificando precisão.

Quando fizer cálculos com floats, você precisa adicionar um ponto decimal, do contrário o número será tratado como um int. Veja a página sobre constantes de ponto flutuante para mais detalhes.

Sintaxe

float var = val

=== Parâmetros var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O trecho de código demonstra algumas operações com floats.

float myfloat;

float sensorCalbrate = 1.117;

int x;

int y;

float z;

x = 1;

y = x / 2; // y agora contém 0, ints não podem guardar frações

z = (float)x / 2.0; // z agora contém 0.5 (você deve usar 2.0, não 2)

int

[Tipos de Dados]

Descrição

Ints (integer ou inteiros) são o tipo o tipo de dados primário para armazenamento de números.

No Arduino Uno (e outras placas baseadas em micros ATmega) um int armazena um valor 16-bit (2 bytes). Isso garante um intervalo de -32,768 a 32,767 (um valor mínimo de -2^15 e um valor máximo de (2^15) - 1). No Arduino Due e outras placas baseadas em micros SAMD (como o MKR1000 e Zero), um int armazena um valor 32-bit (4 bytes). O que garante um intervalo de -2,147,483,648 a 2,147,483,647 (um valor mínimo de -2^31 e um valor máximo de (2^31) - 1).

Variáveis int armazenam números negativos com uma técnica chamada (Complemento de 2). O bit mais significativo, as vezes chamado de o "bit de sinal", indica que o número é negativo. O resto dos bits são invertidos e 1 é adicionado.

O Arduino cuida dos detalhes nas operações com números negativos por você, tal que operações aritméticas funcionam transparentemente na maneira esperada. No entanto, pode-se haver uma complicação ao lidar com o operador de deslocamento de bits a direita (>>).

Sintaxe

int var = val;

=== Parâmetros var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

Esse código cria uma variável do tipo integer chamada 'contador', a qual é inicialmente atribuído o valor 0 (zero). A váriavel incrementa em 1 (um) a cada execução do loop e é mostrada no monitor serial.

int contador = 0; //cria uma variável integer chamada 'contador'

void setup() {

Serial.begin(9600); // usa a porta serial para imprimir o número

}

void loop() {

contador++; // soma 1 ao int contador a cada execução do loop

Serial.println(contador); // Imprime o estado atual do contador

delay(1000);

}

Notas e Advertências

Quando variáveis com sinal são forçadas a exceder sua capacidade máxima ou mínima, elas estouram; ou do Inglês, sofrem overflow. O resultado de um overflow é imprevisível, então isso deve ser evitado. Um sintoma típico de um overflow é a variável "rolar" de sua capacidade máxima para a mínima ou vice-versa, mas não é sempre o caso. Se você deseja esse comportamento, use o tipo unsigned int.

long

[Tipos de Dados]

Descrição

Variáveis long são variáveis de tamanho extendido para armazenamento de números, armazenam 32 bits (4 bytes), de -2,147,483,648 a 2,147,483,647.

Ao se fazer cálculos com inteiros, pelo menos um dos números deve ser seguido por um L, forçando-o a ser um long. Veja a página sobre Constantes Inteiras para detalhes.

Sintaxe

long var = val;

=== Parâmetros var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

long speedOfLight = 186000L; // veja a página sobre constantes inteiras para uma explicação sobre o 'L'

short

[Tipos de Dados]

Descrição

O tipo short é um tipo de dado 16-bit.

Em todos os Arduinos (baseados no ATMega ou ARM) um short armazena um valor 16-bit (2 bytes). Isso fornece um intervalo de -32,768 a 32,767 (valor mínimo de -2^15 e valor máximo de (2^15) - 1).

Sintaxe

short var = val;

=== Parâmetros var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

short ledPin = 13

string

[Tipos de Dados]

Descrição

As strings de texto podem ser representadas de duas maneiras. Você pode usar o tipo de dado String, que é parte do core desde a versão 0019, ou você pode fazer uma string com um vetor do tipo char e terminá-lo em null. Essa página descreve o segundo método. Para mais detalhes sobre o objeto String, que possui mais funcionalidades ao custo de mais memória ocupada, veja a página sobre os objetos String.

Sintaxe

Todos as declarações seguintes são válidas para strings.

char Str1[15];

char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};

char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};

char Str4[] = "arduino";

char Str5[8] = "arduino";

char Str6[15] = "arduino";

Possibilidades para declarar strings

Declarar um vetor de chars sem inicializá-lo, como em Str1

Declarar um vetor de chars (com um char extra) e o compilador irá adicionar o caractere null requerido, como em Str2

Adicionar o caractere null explicitamente, Str3

Inicializar uma string constante em aspas; o compilador irá ajustar o tamanho do vetor para a string mais o caractere null terminador, como em Str4

Inicializar o vetor com o tamanho explícito e a string constante, Str5

Inicializar o vetor, deixando espaço extra para uma string maior, Str6

Terminação em null

Geralmente, strings são terminadas com um caractere null (código ASCII 0). Isso permite que funções (como Serial.print()) saibam onde está o final de uma string. Do contrário, essas funções continuariam lendo bytes consequentes da memória que não fazem parte da string.

Isso significa que sua string precisa ter espaço para mais um caractere além do texto que você quer que ela contenha. Por isso que Str2 e Str5 precisam ter oito carcteres, mesmo que "arduino" seja apenas sete - a última posição é automaticamente preenchida com um caractere null. Str4 é automaticamente ajustada para oito caracteres, sendo um para o null extra. Em Str3, o caractere null foi explicitamente incluído, adicionando-se '\0'.

Note que é possível criar uma string sem uma caractere null no final (ex. se se você tivesse especificado o tamanho de Str2 como sete em vez de oito). Porém isso irá quebrar a maioria das funções que usam strings, então você não deve fazê-lo intencionalmente. Se você notar algo se comportando estranhamente no seu programa (operando em caracteres fora da string), isso pode ser o problema.

Aspas simples ou aspas duplas?

As strings são sempre definidas envoltas em aspas duplas ("Abc") e caracteres são sempre definidos envoltos em aspas simples('A').

Escrevendo strings muito longas

Você pode escrever uma string muito longa em várias linhas, da seguinte forma:

char myString[] = "Essa é a primeira linha"

" Essa é a segunda linha"

" etcetera";

Vetores de strings

Frequentemente é conveniente, quando lidando-se com grandes quantidades de texto, como um projeto com um display LCD, configurar um vetor de strings. Como strings por sí próprias já são vetores, isso é na verdade um exemplo de um vetor bidimensional.

No código abaixo, o asterisco depois do tipo de dado char “char\*” indica que esse é um vetor de "ponteiros". Todos os nomes de vetores são na verdade ponteiros, então isso é necessário para criar um vetor de vetores. Ponteiros são uma partes mais esotéricas da linguagem C++ para inicinates entenderem, mas não é necessário entender ponteiros profundamente para usá-los efetivamente aqui.

Código de Exemplo

char \*myStrings[] = {"This is string 1", "This is string 2", "This is string 3",

"This is string 4", "This is string 5", "This is string 6"

};

void setup() {

Serial.begin(9600);

}

void loop() {

for (int i = 0; i < 6; i++) {

Serial.println(myStrings[i]);

delay(500);

}

}

nsigned char

[Tipos de Dados]

Descrição

Um tipo de dado sem sinal, que ocupa um byte na memória. O mesmo que o tipo de dado byte. O tipo de dado unsigned char armazena valores de 0 a 255.

Para consistência do estilo de programação Arduino, o tipo de dado byte é preferido.

Sintaxe

unsigned char var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O trecho de código abaixo cria uma variável do tipo unsigned char

unsigned char myChar = 240;

unsigned int

[Tipos de Dados]

Descrição

No UNO e outras placas baseadas nos microcontroladores ATmega, unsigned ints (inteiros sem sinal) são o mesmo que o tipo int no que esses também armazenam um valor de 2 bytes. Em vez de guardar números negativos, no entanto, esses apenas armazenam valores positivos, garantindo um intervalo útil de 0 a 65,535 ((2^16) - 1).

O Due e outras placas ARM armazenam um unsigned int em 4 bytes (32 bits), de 0 a 4,294,967,295 (2^32 - 1).

A diferença entre unsigned ints e ints (com sinal), está na forma como o bit mais significativo, as vezes chamado de o bit "de sinal", é interpretado. No tipo int do Arduino (que possui sinal), se o bit mais significativo é "1", o número é interpretado como um número negativo, e os outros 15 bits são interpretados com (complemento de 2).

Sintaxe

unsigned int var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O trecho de código abaixo cria uma variável unsigned int chamada ledPin e a atribui o valor 13.

unsigned int ledPin = 13;

Notas e Advertências

Quando variáveis unsigned tem sua capacidade máxima excedida, elas "estouram" e voltam para 0, e também o contrário. Exemplo:

unsigned int x;

x = 0;

x = x - 1; // x agora contém 65535

x = x + 1; // x agora contém 0

A matemática com variávies unsigned pode produzir resultados inesperados, mesmo se a variável unsigned nunca estourar.

O microcontrolador aplica as seguintes regras:

O cálculo é feito de acordo com a variável de destino. Exemplo, se a variável destino possui sinal, ele irá realizar o cálculo levando em conta o sinal, mesmo que ambas as variáveis de entrada sejam unsigned.

No entanto, em um cálculo que requer um resultado intermediário, o escopo da variável intermediária não é especificado pelo código. Nesse caso, o micro irá usar matemática sem sinal para o resultado intermediário, porque ambas as entradas são unsigned!

unsigned int x = 5;

unsigned int y = 10;

int resultado;

resultado = x - y; // 5 - 10 = -5, como esperado

resultado = (x - y) / 2; // 5 - 10 com matemática unsigned é 65530! 65530/2 = 32765

// solução: usar variáveis com sinal, ou fazer o cálculo passo a passo.

resultado = x - y; // 5 - 10 = -5, como esperado

resultado = resultado / 2; // -5/2 = -2 (matemática inteira, casas decimais não são consideradas)

Então porque usar variáveis sem sinal?

Quando o comportamento de estouro é desejado, ex. contadores

A variável com sinal é muito pequena, e se quer evitar a perda de memória e performance de long/float.

unsigned long

[Tipos de Dados]

Descrição

Variáveis unsigned long são variáveis de tamanho extendido para armazenamento de números, que armazenam 32 bits (4 bytes). Diferentemente de longs padrão, unsigned longs não guardam números negativos, o que faz com que possam armazenar valores de 0 a 4,294,967,295 (2^32 - 1).

Sintaxe

unsigned long var = val;

=== Parâmetros var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O código abaixo cria uma variável do tipo long para guardar o número de milissegundos desde que o programa iniciou.

unsigned long time; //variável do tipo long

void setup() {

Serial.begin(9600);

}

void loop() {

Serial.print("Time: ");

time = millis();

//imprime o tempo desde que o programa iniciou, em milissegundos

Serial.println(time);

// espera um segundo, para não enviar quantidades massivas de dados

delay(1000);

}

vetor

[Tipos de Dados]

Descrição

Um vetor (array) é uma coleção de variáveis que são acessadas com um número índice. Vetores na linguagem C++, na qual o Arduino é baseado, podem ser complicados, mas usar vetores simples é relativamente fácil.

Criando-se um Vetor

Todos os métodos abaixo são formas válidas de criar (declarar) um vetor.

int meusInts[6];

int meusPinos[] = {2, 4, 8, 3, 6};

int meusValSens[6] = {2, 4, -8, 3, 2};

char mensagem[6] = "hello";

Você pode declarar um vetor sem inicializá-lo, como em meusInts.

Em meusPinos, nós declaramos um vetor sem explicitamente escolher um tamanho. O compilador conta os elementos e cria um vetor de tamanho apropriado.

Finalmente você pode inicializar e também especificar o tamanho do vetor, como em meusValSens. Note que quando declarar um vetor do tipo char, um elemento a mais que sua inicialização é requerido, para armazenar o caractere null necessário.

Acessando-se um Vetor

Vetores são indexados a partir de 0, isto é, referindo-se a inicialização dos vetores acima, o primeiro elemento do vetor está no índice 0, então

meusValSens[0] == 2, meusValSens[1] == 4, e assim por diante.

Isso também significa que em um vetor com dez elementos, o índice nove é o último elemento. Então:

int meuVetor[10] = {9, 3, 2, 4, 3, 2, 7, 8, 9, 11};

// meuVetor[9] contém 11

// meuVetor[10] é inválido e contém informação aleatória (endereço de memória fora do vetor)

Por esse motivo, você deve ser cuidadoso ao acessar vetores. Acessar um elemento depois do final de um vetor (usar um índice maior que o tamanho declarado - 1) é ler de uma parte da memória que pode estar em uso para outros propósitos. Ler desses locais provavelmente não vai fazer mais que retornar dados inválidos. Escrever em locais aleatórios da memória é definitivamente uma má ideia e pode frequentemente levar a péssimos resultados como crashes ou mal funcionamento do programa. Isso pode também ser um bug difícil de encontrar.

Diferentemente de BASIC ou JAVA, o compilador C++ não faz checagem para confirmar se o acesso a um elemento está dentro dos limites do tamanho declarado do vetor.

Para atribuir um valor a um elemento do vetor:

meusValSens[0] = 10;

Para ler um elemento de um vetor:

x = mySensVals[4];

Vetores e loops For

Vetores são frequentemente manipulados dentro de loops for, onde o contador do loop é usado como o índice para cada elemento. Por exemplo, para imprimir os elementos de um vetor através da porta serial, você pode fazer algo do tipo:

for (byte i = 0; i < 5; i = i + 1) {

Serial.println(meusPinos[i]);

}

Código de Exemplo

Para um programa completo que demonstra o uso de vetores, veja o exemplo da Super Máquina dos Tutoriais Arduino (Em Inglês).

void

[Tipos de Dados]

Descrição

A palavra chave void é usada apenas em declarações de funções. Ela indica que é esperado que a função não retorne nenhuma informação para a função da qual foi chamada.

Código de Exemplo

O código mostra como usar a palavra chave void.

// são realizadas ações nas funções "setup" e "loop"

// porém não é retornada nenhuma informação quando as mesmas são executadas

void setup() {

// ...

}

void loop() {

// ...

}

word

[Tipos de Dados]

Descrição

Uma variável word armazena um número sem sinal de ao menos 16 bits, de 0 A 65535.

Sintaxe

word var = val;

Parâmetros

var: nome da variável

val: valor a ser atribuído à variável

Código de Exemplo

O trecho de código abaixo cria uma variável do tipo word.

word w = 10000;

Escopo de Variáveis e Qualificadores

const

[Escopo de Variáveis e Qualificadores]

Descrição

A palavra-chave const é uma abreviação de constante. É um qualificador de variáveis que modifica o comportamento da variável, fazendo com que a variável seja de "apenas-leitura". Isso significa que a variável pode ser usada como qualquer outra variável de seu tipo, mas seu valor não pode ser mudado. Você terá um erro de compilação se tentar atribuir um valor a uma variável const.

Constantes definidas com a palavra chave const obedecem as regras dos escopos de variáveis que governam variáveis. Isso, e as armadilhas de se usar #define, fazem com que o modificador const um método superior de se definir constantes e é favorecido em lugar de se usar #define.

Código de Exemplo

const float pi = 3.14;

float x;

// ....

x = pi \* 2; // constantes podem ser usadas em cálculos

pi = 7; // ilegal - você não pode modificar o valor de uma constante

Notas e Advertências

#define ou const

Você pode usar ambos const ou #define para criar constantes numéricas ou strings constantes. Para vetores, você irá precisar usar const. Em geral, const é favorecido em lugar de #define ao se definir constantes.

escopo

[Escopo de Variáveis e Qualificadores]

Descrição

Variáveis na linguagem de programação C++, a qual o Arduino usa, possuem uma propriedade chamada escopo. Isso contrasta com antigas versões de linguagens como o BASIC onde toda variável era uma variável global.

Uma variável global é uma que pode ser vista por todas as funções de um programa. Diferentemente, variáveis locais são visíveis apenas às funções nas quais são declaradas. No ambiente Arduino, qualquer variável declarada fora de uma função (ex. setup(), loop(), etc. ), é uma variável \_ global\_.

Quando programas começam a ficar muito longos e complexos, variáveis locais são uma forma útil de garantir que apenas uma função tenha acesso as suas próprias variáveis. Isso previne erros de programação quando uma função inadvertidamente modifica variáveis usadas por outra função.

Às vezes também é vantajoso declarar e inicializar uma variável dentro de um loop for. Isso cria uma variável que pode ser acessada apenas dentro do próprio bloco do loop for.

Código de Exemplo

int gPWMval; // qualquer função poderá acessar essa variável

void setup() {

// ...

}

void loop() {

int i; // "i" é "visível" apenas dentro de "loop"

float f; // "f" é "visível" apenas dentro de "loop"

// ...

for (int j = 0; j < 100; j++) {

// a variável j pode ser acessada apenas dentro das chaves do loop for

}

}

volatile

[Escopo de Variáveis e Qualificadores]

Descrição

A palavra chave volatile é conhecida como um qualificador de variáveis, e é geralmente usada antes do tipo de dado da variável, para modificar a forma com qual o compilador e o programa tratam a variável.

Declarar uma variável como volatile é uma diretiva para o compilador. O compilador é o software que traduz seu código C/C++ em linguagem de máquina, que são as instruções reais entendidas pelo chip microcontrolador na placa Arduino.

Especificamente, essa diretiva faz com que o compilador carregue a variável da RAM e não de um registrador, que é uma localização temporária da memória onde variáveis do programa são guardadas e manipuladas. Sob certas circunstâncias, o valor de uma variável armazenada em registradores pode ser impreciso.

Uma variável deve ser declarada volatile quando seu valor pode ser mudado por algo além do controle da seção de código na qual ela aparece, como uma thread paralela. No Arduino, o único lugar do código onde isso pode ocorrer é em seções associadas com interrupções, chamadas de rotina de serviço de interrupção (interrupt service routine).

variáveis volatile int ou long

Se a variável volatile é maior que um byte (ex. um int de 16 bits ou long de 32 bits), o microcontrolador não é capaz de lê-la em apenas um passo, porquê é um microcontrolador 8 bits. Isso significa que enquanto a seção principal do seu código (ex. seu loop) lê o primeiro ou os primeiros bytes da variável, a interrupção pode mudar os outros bytes. Isso irá produzir valores inesperados na variável.

Solução:

Enquanto a variável é lida, as interrupções são desativadas, de forma que estas não podem bagunçar as variáveis durante a leitura. Há algumas formas de se fazer isso:

noInterrupts

usar a macro ATOMIC\_BLOCK. Operações atômicas são operações únicas para o microcontrolador - a menor unidade possível.

Código de Exemplo

// muda o estado do LED quando o estado do botão é mudado

int pin = 13;

volatile int state = LOW;

void setup() {

pinMode(pin, OUTPUT);

attachInterrupt(digitalPinToInterrupt(2), blink, CHANGE);

}

void loop() {

digitalWrite(pin, state);

}

void blink() {

state = !state;

}

#include <util/atomic.h> // essa biblioteca inclui a macro ATOMIC\_BLOCK.

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE) {

// código com interrupções bloqueadas (operações atômicas consecutivas não irão ser interrompidas)

}

Utilitários

PROGMEM

[Utilitários]

Descrição

Armazena dados na memória flash (memória de programa) em vez da SRAM. Uma descrição dos vários tipos de memória das placas Arduino pode ser encontrada (Em Inglês) nessa página.

A palavra-chave PROGMEM é um modificador de variáveis, que pode ser usada apenas com os modificadores de variáveis definidos em pgmspace.h. Ela diz ao compilador "armazene essa informação na memória flash", em vez da SRAM, onde seria normalmente armazenada.

PROGMEM é parte da biblioteca pgmspace.h. Essa biblioteca é incluída automaticamente em versões modernas da IDE, porém se você estiver usando uma versão da IDE anterior à 1.0 (2011), iŕa precisar incluir ela no topo do seu sketch, da seguinte forma:

#include <avr/pgmspace.h>

Sintaxe

const dataType variableName[] PROGMEM = {data0, data1, data3…​};

Onde:

dataType - qualquer tipo de dados

variableName - o nome do seu vetor de dados

Note que porque PROGMEM é um modificador de variável, não há uma regra rigorosa de onde ele deve ir, então o compilador aceita todos os tipos de definição abaixo, que também são sinônimos. Mesmo assim, experimentos indicaram que, em várias versões do Arduino (relacionado a versão do GCC), PROGMEM pode funcionar em uma localização e não em outra. O exemplo da "tabela de strings" abaixo foi testado com a versão 13 do Arduino. Versões mais antigas da IDE podem funcionar melhor se PROGMEM for incluído depois do nome da variável.

const dataType variableName[] PROGMEM = {}; // use essa forma

const PROGMEM dataType variableName[] = {}; // ou essa

const dataType PROGMEM variableName[] = {}; // mas não essa

Enquanto PROGMEM pode ser usada em uma única variável, realmente só vale a pena o trabalho de usá-lo se você tiver um bloco de dados maior para ser armazenado, o que geralmente é mais fácil de fazer com vetores (ou outra estrutura de dados da linguagem C++ fora do escopo da nossa discussão atual).

Usar PROGMEM é também um procedimento realizado em dois passos. Depois de resgatar os dados da memória flash, são requeridas funções especiais, também definidas na biblioteca pgmspace.h, para ler os dados da memória de programa para a memória SRAM, tal que possamos fazer algo com eles.

Código de Exemplo

Os fragmentos de código abaixo ilustram como ler e escrever unsigned chars (bytes) e ints (2 bytes) com PROGMEM.

// armazena alguns unsigned ints

const PROGMEM uint16\_t conjunto[] = {65000, 32796, 16843, 10, 11234};

// armazena alguns chars

const char mensagem[] PROGMEM = {"Um pequeno jabuti xereta viu dez cegonhas felizes"};

unsigned int displayInt;

char meuChar;

void setup() {

Serial.begin(9600);

while (!Serial); // Espera a porta serial conectar. Necessário para placas com USB nativa

// Lê da memória flash um int (2-bytes, ou word)

for (byte k = 0; k < 5; k++) {

displayInt = pgm\_read\_word\_near(conjunto + k);

Serial.println(displayInt);

}

Serial.println();

// Lê um caractere da flash

for (byte k = 0; k < strlen\_P(mensagem); k++) {

meuCHar = pgm\_read\_byte\_near(mensagem + k);

Serial.print(meuCHar);

}

Serial.println();

}

void loop() {

//Nada aqui

}

Vetores de strings

É frequentemente conveniente, quando se trabalha com grandes quantidade de texto, como um projeto com display LCD, por exemplo, preparar vetores de strings. Porque strings por si próprias já são vetores, este é genuinamente um exemplo de um vetor bidimensional.

Essas estruturas tendem a ser muito grandes, de forma que colocá-las na memória pode ser desejável. O código abaixo ilustra a ideia.

/\*

demo com strings e PROGMEM

Como armazenar uma tabela de strings na memória de programa (flash),

e recuperá-los.

Informação resumida de:

http://www.nongnu.org/avr-libc/user-manual/pgmspace.html

Preparar uma tabela (vetor) de strings na memória de programa é relativamente complicado,

mas esse template pode ser seguido.

Preparar as strings é um processo em dois passos. O primeiro é definir as strings.

\*/

#include <avr/pgmspace.h>

const char string\_0[] PROGMEM = "String 0"; // "String 0" etc são as strings a serem armazenadas - adapte ao seu programa.

const char string\_1[] PROGMEM = "String 1";

const char string\_2[] PROGMEM = "String 2";

const char string\_3[] PROGMEM = "String 3";

const char string\_4[] PROGMEM = "String 4";

const char string\_5[] PROGMEM = "String 5";

// Então crie uma tabela para apontar para as suas strings.

const char \*const string\_table[] PROGMEM = {string\_0, string\_1, string\_2, string\_3, string\_4, string\_5};

char buffer[30]; // Tenha certeza que esse buffer é grande o suficiente para armazenar a maior string

void setup() {

Serial.begin(9600);

while (!Serial); // Espera a porta serial conectar. Necessário para placas com USB nativa

Serial.println("OK");

}

void loop() {

/\* Usar a tabela de strings da memória de programa requer o uso de funções especiais para recuperar os dados.

A função strcpy\_P copia uma string do espaço de programa para uma string na RAM (em um "buffer").

Tenha certeza que o seu buffer na RAM é grande o suficiente para armazenar o que quer que seja

que você estiver lendo da memória de programa. \*/

for (int i = 0; i < 6; i++) {

strcpy\_P(buffer, (char \*)pgm\_read\_word(&(string\_table[i]))); // Casts e desreferência necessários, apenas copie.

Serial.println(buffer);

delay(500);

}

}

Notas e Advertências

Por favor note que as variáveis devem ser definidas globalmente OU com a palavra-chave static, para funcionarem com PROGMEM.

O código a seguir não funcionará dentro de uma função:

const char long\_str[] PROGMEM = "Olá, gostaria de falar um pouco sobre mim mesmo.\n";

O código a seguir irá funcionar, mesmo se definido localmente dentro de uma função:

const static char long\_str[] PROGMEM = "Olá, gostaria de falar um pouco sobre mim mesmo.\n";

A macro F()

Quando uma instrução do tipo:

Serial.print("Escreve algo no Monitor Serial");

é usada, a string a ser impressa é normalmente salva na RAM. Se o seu sketch imprime muita coisa no Monitor Serial, você pode facilmente encher a RAM. Se você tiver espaço livre na memória flash, você pode facilmente indicar ao compilador que a string deve ser salva na FLASH usando a sintaxe:

Serial.print(F("Esse texto foi guardado na memória flash do Arduino"));

sizeof()

[Utilitários]

Descrição

O operador sizeof retorna o número de bytes de uma variável, ou o número de bytes ocupados em um vetor.

Sintaxe

sizeof(variável)

Parâmetros

variável: uma variável ou vetor de qualquer tipo (ex: int, float, byte)

Retorna

O número de bytes em uma variável ou o número de bytes ocupados em um vetor. (size\_t)

Código de Exemplo

O perador sizeof é útil para manipular arrays (tais como strings), onde é conveniente ser possível mudar o tamanho do vetor sem quebrar outras partes do programa.

O programa abaixo imprime um string um caractere de cada vez. Tente mudar o texto da frase.

char minhaStr[] = "Esse é um teste";

void setup() {

Serial.begin(9600);

}

void loop() {

for (byte i = 0; i < sizeof(minhaStr) - 1; i++) {

Serial.print(i, DEC);

Serial.print(" = ");

Serial.write(minhaStr[i]);

Serial.println();

}

delay(5000); // espera 5 segundos

}

Notas e Advertências

Note que sizeof retorna o número total de bytes. Então, para vetores de tipos de dados maiores, como int, o loop for parece algo do tipo abaixo.

int meusValores[] = {123, 456, 789};

// this for loop works correctly with an array of any type or size

for (byte i = 0; i < (sizeof(meusValores)/sizeof(meusValores[0])); i++) {

// fazer algo com meusValores[i]

}

ESTRUTURAS

Os elementos da linguagem Arduino (C++).

Sketch

loop()

[Sketch]

Descrição

Depois de criar uma função setup(), a qual inicializa e atribui os valores iniciais, a função loop() faz precisamente o que o seu nome sugere, e repete-se consecutivamente enaqunto a placa estiver ligada, permitindo o seu programa mudar e responder a essas mudanças. Use-a para controlar ativamente uma placa Arduino.

Código de Exemplo

int buttonPin = 3;

// setup inicializa a porta serial e o pino para o botão

void setup() {

Serial.begin(9600);

pinMode(buttonPin, INPUT);

}

// loop checa o estado do botão repetidamente, e envia

// pela serial um 'H' se este está sendo pressionado

void loop() {

if (digitalRead(buttonPin) == HIGH) {

Serial.write('H');

}

else {

Serial.write('L');

}

delay(1000);

}

setup()

[Sketch]

Descrição

A função setup() é chamada quando um sketch inicia. Use-a para inicializar variáveis, configurar o modo dos pinos(INPUT ou OUTPUT), inicializar bibliotecas, etc. A função setup() será executada apenas uma vez, apoós a placa ser alimentada ou acontecer um reset.

Código de Exemplo

int buttonPin = 3;

void setup() {

// Inicializa a porta serial

Serial.begin(9600);

// configura o pino 3 como INPUT

pinMode(buttonPin, INPUT);

}

void loop() {

// ...

}

break

[Estruturas de Controle]

Descrição

break é usado usado para sair de um laço for, while ou do…​while, ignorando a condição padrão do loop. Também é usada para sair do comando switch case.

Código de Exemplo

No códgo seguinte, o break quebra o loop for quando o valor do sensor excede o limiar.

int lim = 40;

for (int x = 0; x < 255; x ++) {

analogWrite(PWMpin, x);

sens = analogRead(sensorPin);

if (sens > lim) { // "foge" do o loop `for`

x = 0;

break;

}

delay(50);

}

continue

[Estruturas de Controle]

Descrição

O comando continue "pula" o resto da iteração atual de um loop (for, while, ou do…​while). Esse comando continua a checar a expressão condicional do loop, e procede com qualquer iterações subsequentes.

Código de Exemplo

O código abaixo escreve o valor de 0 a 255 ao pino PWMpin, mas pula os valores no intervalo 41 a 119.

for (int x = 0; x <= 255; x ++) {

if (x > 40 && x < 120) { // cria um salto nos valores

continue;

}

analogWrite(PWMpin, x);

delay(50);

}

do...while

[Estruturas de Controle]

Descrição

O loop do…​while funciona da mesma forma que o loop while, com a exceção de a condição ser testada no final do loop, tal que o loop será executado pelo menos uma vez.

Sintaxe

do {

// bloco de comandos

} while (condição);

A condição é uma expressão booleana que é avaliada como verdadeiro ou falso, respectivamente true ou false na linguagem Arduino.

Código de Exemplo

int x = 0;

do {

delay(50); // espera os sensores estabilizarem

x = readSensors(); // checa os sensores

} while (x < 100);

continue

[Estruturas de Controle]

Descrição

O comando continue "pula" o resto da iteração atual de um loop (for, while, ou do…​while). Esse comando continua a checar a expressão condicional do loop, e procede com qualquer iterações subsequentes.

Código de Exemplo

O código abaixo escreve o valor de 0 a 255 ao pino PWMpin, mas pula os valores no intervalo 41 a 119.

for (int x = 0; x <= 255; x ++) {

if (x > 40 && x < 120) { // cria um salto nos valores

continue;

}

analogWrite(PWMpin, x);

delay(50);

}

else

[Estruturas de Controle]

Descrição

A combinação if…​else permite maior controle sobre o fluxo de código que o comando mais básico if, por permitir múltiplos testes serem agrupados juntos. Uma cláusula else (se presente) será executada se a condição do comando if resulta em false. O else pode proceder outro teste if, tal que múltiplos, testes mutualmente exclusivos podem ser executados ao mesmo tempo.

Cada teste irá proceder para o próximo até um teste que resulte em verdadeiro seja encontrado. Quando esse teste for encontrado, o bloco de código correspondente a condição é executado, e o programa então continua a partir da linha seguinte a estrutura if-else completa. Se nenhum teste resultar em verdadeiro, o bloco else padrão é executado, se um estiver presente, que dita o comportamento padrão.

Note que um bloco else if pode ser usado sem um bloco else no final e vice-versa. Um número praticamente ilimitado de blocos else if conectados é permitido.

Sintaxe

if (condição1) {

// faz coisa A

}

else if (condição2) {

// faz coisa B

}

else {

// faz coisa C

}

Código de Exemplo

Abaixo um trecho de código de um sistema de controle de temperatura

if (temperatura >= 70) {

//Perigo! Desligar o sistema

}

else if (temperatura >= 60 && temperatura < 70) {

//Cuidado! Requerida a atenção do usuário

}

else {

//Seguro! Continue as tarefas usuais...

}

for

[Estruturas de Controle]

Descrição

O comando for [e usado para repetir um bloco de código envolvido por chaves. Um contador de incremento é geralmente utilizado para terminar o loop. O comando for é útil para qualquer operação repetitiva, e é usado frequentemente com vetores para operar em coleções de dados ou pinos.

Sintaxe

for (inicialização; condição; incremento) {

//comando(s);

}

A inicialização ocorre primeiro e apenas uma vez. A cada repetição do loop, a condição é testada; se é verdadeira (true), o bloco de comandos, e o incremento são executados. Quando a condição se torna falsa (false), o loop termina.

Código de Exemplo

// Varia o brilho de um LED usando um pino PWM

int pinoPWM = 10; // LED em série com um resistor de 470 ohm no pino 10

void setup() {

// setup não necessário

}

void loop() {

for (int i = 0; i <= 255; i++) {

analogWrite(pinoPWM, i);

delay(10);

}

}

Notas e Advertências

O loop for na linguagem C é muito mais flexível que os loops `for` encontrados em outras linguagens. Quaisquer dos três elementos da sintaxe podem ser omitidos, porém os ponto e vírgula (';') são necessários. Além disso, os comandos para inicialização, condição e incremento podem ser quaisquer comandos válidos na linguagem C, mesmo com variáveis não relacionadas ao loop, e podem usar quaisquer tipos de dados da linguagem, incluindo floats. Esses tipos de comandos for incomuns podem prover soluções rápidas para alguns problemas raros de programação.

Por exemplo, usar uma multiplicação no comando de incremento irá gerar uma progressão logarítmica:

for (int x = 2; x < 100; x = x \* 1.5) {

println(x);

}

Gera: 2,3,4,6,9,13,19,28,42,63,94.

Outro exemplo, Aplica um efeito de fading crescente e decrescente em um LED com apenas um loop for:

void loop() {

int x = 1;

for (int i = 0; i > -1; i = i + x) {

analogWrite(pinoPWM, i);

if (i == 255) {

x = -1; // muda a direção no pico

}

delay(10);

}

}

goto

[Estruturas de Controle]

Descrição

Transfere o fluxo do programa para um ponto rotulado no código.

Sintaxe

rótulo:

goto rótulo; // envia o fluxo do programa de volta para o rótulo

Código de Exemplo

O código abaixo "foge" de dentro de todos os laços caso a leitura no pino analógico 0 seja maior que 250.

for (byte r = 0; r < 255; r++) {

for (byte g = 255; g > 0; g--) {

for (byte b = 0; b < 255; b++) {

if (analogRead(0) > 250) {

goto cancelar;

}

// mais comandos ...

}

}

}

cancelar:

// mais comandos ...

Notas e Advertências

O uso do goto é desencorajado na programação em C++, e alguns autores de livros afirmam que o uso do comando goto nunca é necessário, mas se usado judiciosamente ele pode simplificar certos programas. A pela qual muitos programadores desaprovam o uso do goto é que com o uso descuidado de comando goto, é fácil criar um programa com fluxo indefinido, que não pode ser depurado.

Tendo dito isso, há algumas instâncias onde o comando goto pode ser considerado, por simplificar o código. Uma dessas situações é sair de laços for profundamente aninhados, ou blocos if, em determinada condição.

if

[Estruturas de Controle]

Descrição

O comando if checa uma condição e executas o comando a seguir ou um bloco de comandos delimitados por chaves, se a condição é verdadeira ('true').

Sintaxe

if (condição) {

//comando(s)

}

Parâmetros

condição: uma expressão booleana, isto é, que pode resultar apenas em true ou false

Código de Exemplo

As chaves podem ser omitidas depois de um comando if. Se isso é feito, a próxima linha (definida pelo ponto e vírgula) é interpretada como o único comando condicional. Para mais de um comando, use as chaves para delimitar o bloco de comandos.

if (x > 120) {

digitalWrite(pinoLED, HIGH);

}

if (x > 120) {

digitalWrite(pinoLED, HIGH);

}

if (x > 120) {

digitalWrite(pinoLED, HIGH);

}

if (x > 120) {

digitalWrite(pinoLED1, HIGH);

digitalWrite(pinoLED2, HIGH);

} // todas as formas acima estão corretas

Notas e Advertências

As expressões sendo testadas dentro dos parênteses geralmente requerem o uso de um ou mais dos operadores mostrados abaixo.

Operadores de comparação:

x == y (x é igual a y)

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)

Cuidado para não usar acidentalmente o símbolo de igual simples (ex. if (x = 10) ). O símbolo de igual simples é o operador de atribuição, se atribui 10 a x (coloca o valor 10 na variável x). Em vez disso, o símbolo de igual duplo (ex. if (x == 10) ) deve ser usado, que é o operador de comparação, e testa se x é igaul a 10 ou não. O último é apenas verdadeiro se x é igual a 10, enquanto o primeiro comando mostrado sempre resultará em verdadeiro.

Isso acontece porque a linguagem C++ interpreta if (x=10) da seguinte forma: 10 é atribuído a x (Lembre que o símbolo de igual simples é o (operador de atribuição)), então x agora contém 10. então o comando 'if' testa 10, o que sempre resulta true, desde que qualquer número diferente de zero resulta em true. Consequentemente, if (x = 10) irá sempre resultar em true, o que não é desejável ao se usar um comando 'if'. Além disso, a variável x irá receber o valor 10, o que também é indesejado.

return

[Estruturas de Controle]

Descrição

Termina uma função e retorna um valor, caso desejado.

Sintaxe

return;

return valor; // ambas as formas são válidas

Parâmetros

valor: qualquer variável ou constante de qualquer tipo de dado

Código de Exemplo

Uma função para comparar a saída de um sensor com um limiar

int checaSensor() {

if (analogRead(0) > 400) {

return 1;

}

else {

return 0;

}

}

The return keyword is handy to test a section of code without having to "comment out" large sections of possibly buggy code.

void loop() {

return;

// the rest of a dysfunctional sketch here

// this code will never be executed

}

switch...case

[Estruturas de Controle]

Descrição

Da mesma forma que o comando if, o comando switch case controla o fluxo do programa permitindo ao programador especificar código diferente para ser executado em várias condições. Em particular, um comando switch compara o valor de uma variável aos valores especificados nos comandos case. Quando um comando case é encontrado cujo valor é igual ao da variável, o código para esse comando case é executado.

A palavra-chave break interrompe o comando switch, e é tipicamente usada no final de cada case. Sem um comando break, o comando switch irá continuar a executar as expressões seguintes (desnecessariamente) até encontrar um break, ou até o final do comando switch ser alcançado.

Sintaxe

switch (var) {

case valor1:

// comando(s)

break;

case valor2:

// comando(s)

break;

default:

// comando(s)

break;

}

Parâmetros

var: uma variável para ser comparada com os vários cases. Tipos de dados permitidos: int, char

valor1, valor2: constantes. Tipos de dados permitidos: int, char

Retorna

Nada

Código de Exemplo

switch (var) {

case 1:

// faz algo quando var é igual a 1

break;

case 2:

// faz algo quando var é igual a 1

break;

default:

// Se nenhum dos anteriores, faz o caso padrão default

// default é opcional

break;

}

while

[Estruturas de Controle]

Descrição

Um loop while irá se repetir continuamente, e infinitamente, até a expressão dentro dos parênteses (), se torne falsa. Algo deve mudar a variável testada, ou o loop while nunca irá encerrar. Isso pode ser no seu código, por exemplo, uma variável incrementada, ou uma condição externa, como a leitura de um sensor.

Sintaxe

while (condição) {

// código a ser executado repetidamente

}

A condição é uma expressão booleana que resulta em true ou false.

Código de Exemplo

var = 0;

while (var < 200) {

// faz algo repetitivo 200 vezes

var++;

}

Outros Elementos da Sintaxe

#define

[Outros Elementos da Sintaxe]

Descrição

#define é uma diretiva muito útil da linguagem C++ que permite ao programador dar um nome a um valor constante antes de o programa ser compilado. Constantes definidas no arduino não ocupam nenhum espaço na memória de programa do chip. O compilador irá substituir referências a essas constantes pelo valor definido no tempo de compilação.

Isso pode ter alguns efeitos colaterais desagradáveis no entanto, por exemplo, se o nome de uma constante que foi definida com #defined é incluído em outra constante ou nome de uma variável. Nesse caso o texto seria trocado pelo número (ou texto) definido com #define.

Em geral, a palavra-chave const é recomendada para se definir constantes e deveria ser usada em vez de #define.

Sintaxe

#define nomeDaConstante valor

Note que o # é necessário.

Código de Exemplo

#define pinoLED 3

// O compilador irá substituir qualquer menção de pinoLED com o valor 3 no tempo de compilação.

Notas e Advertências

Não há ponto e vírgula após a diretiva #define. Se você incluir uma, o compilador irá acusar erros.

#define pinoLED 3; // isso é inválido

Similarmente, incluir sinal de igual após #define também resultará em erros

#define pinoLED = 3 // também é inválido

#include

[Outros Elementos da Sintaxe]

Descrição

A diretiva #include é usada para incluir bibliotecas externas ao seu sketch. Isso dá acesso a um grande número de bibliotecas padrão da linguagem C (grupos de funções prontas), e também bibliotecas escritas especialmente para a linguagem Arduino.

A referência principal para bibliotecas C AVR C (AVR é a referência dos chips Microchip (previamente Atmel) utilizados em muitas placas Arduino) pode ser encontrada aqui.

Note que #include, similarmente a #define, não é terminada com um ponto e vírgula, e o compilador irá mostrar mensagens de erro enigmáticas se você adicionar um.

Código de Exemplo

Esse exemplo inclui a bilbioteca Servo para que suas funções possam ser usadas para controlar um servomotor.

#include <Servo.h>

Servo meuservo; // cria um objeto do tipo Servo para controlar um servomotor

void setup() {

meuservo.attach(9); // associa o motor no pino 9 ao objeto meuservo

}

void loop() {

for (int pos = 0; pos <= 180; pos += 1) { // vai de 0 graus a 180 graus

// em passos de 1 grau

meuservo.write(pos); // diz ao servo para ir para a posição na variável 'pos'

delay(15); // espera 15ms para que o servo chegue a posição

}

for (int pos = 180; pos >= 0; pos -= 1) { // // vai de 180 graus a 0 graus

meuservo.write(pos); // diz ao servo para ir para a posição na variável 'pos'

delay(15); // espera 15ms para que o servo chegue a posição

}

}

/\* \*/

[Outros Elementos da Sintaxe]

Descrição

Comentários são textos no programa que são usadas para informar você e a outros a forma como o programa funciona. Eles são ignorados pelo compilador, e não fazem parte do arquivo gravado no chip, então não ocupam nenhum espaço na memória flash do microcontrolador. O propósito dos comentários lhe ajudar a entender (ou lembrar) como funcionam partes do seu código, ou informar a outros também como o seu programa funciona.

O começo de um comentário em bloco ou comentário de múltiplas linhas é marcado pelo símbolo /\* e o símbolo \*/ marca o seu final. Esse tipo de comentário é chamado assim pois pode se estender por mais de uma linha; um vez que o compilador encontre o símbolo /\*, ele ignora o texto seguinte até encontrar um \*/.

Código de Exemplo

/\* Esse é um comentário válido \*/

/\*

Blink

Acende um LED por um segundo, depois apaga por um segundo, repetidamente.

Esse código encontra-se um domínio público.

(outro comentário válido)

\*/

/\*

if (gwb == 0) { // Comentários de única linha não permitidos dentro de um comentário em bloco

x = 3; /\* Mas outro comentário de múltiplas linhas, não. Esse comentário é inválido \*/

}

// Não esqueça o símbolo para "fechar" o comentário - deve estar balanceado!

\*/

Notas e Advertências

Quando experimentar com código, "comentar" partes de seu programa é uma forma conveniente de remover partes problemáticas do seu código tempoariamente. Isso mantém as linhas de código, mas as transforma em comentários, se forma que o compilador apenas as ignora. Isso pode ser útil quando você estiver procurando um problema, ou quando um programa se recusa a compilar e o erro de compilação é enigmático ou inútil.

//

[Outros Elementos da Sintaxe]

Descrição

Comentários são textos no programa que são usadas para informar você e a outros a forma como o programa funciona. Eles são ignorados pelo compilador, e não fazem parte do arquivo gravado no chip, então não ocupam nenhum espaço na memória flash do microcontrolador. O propósito dos comentários é lhe ajudar a entender (ou lembrar) como funcionam partes do seu código, ou informar a outros também como o seu programa funciona.

Um comentário de uma só linha começa com // (duas barras adjacentes). Esse tipo de comentário termina automaticamente no final da linha. O que quer que seja que estiver após // até o final da linha será ignorado pelo compilador.

Código de Exemplo

Há duas formas diferentes de se usar um comentário de uma só linha:

// O pino 13 tem um LED conectado na maioria das placas Arduino.

// Dá um nome para esse LED:

int led = 13;

digitalWrite(led, HIGH); // acendo o LED (HIGH é o nível da tensão)

Notas e Advertências

Quando experimentar com código, "comentar" partes de seu programa é uma forma conveniente de remover partes problemáticas do seu código temporariamente. Isso mantém as linhas de código, mas as transforma em comentários, se forma que o compilador apenas as ignora. Isso pode ser útil quando você estiver procurando um problema, ou quando um programa se recusa a compilar e o erro de compilação é enigmático ou inútil.

;

[Outros Elementos da Sintaxe]

Descrição

Usado para encerrar um comando.

Código de Exemplo

int a = 13;

Notas e Advertências

Se esquecer de encerar uma linha com um ponto e vírgula irá resultar em um erro de compilação. A mensagem de erro pode ser óbvia, e mencionar a falta de um ponto e vírgula, mas pode também não mencionar a falta do mesmo. Se um erro de compilação incompreensível ou aparentemente ilógico aparecer, uma das primeiras coisas a se checar é a falta de um ponto e vírgula no código anterior a linha mencionada na mensagem de erro do compilador.

{}

[Outros Elementos da Sintaxe]

Descrição

As chaves são uma característica importante da linguagem C++. Elas são usadas em diversas estruturas diferentes, mostradas abaixo, e isso pode as vezes ser confuso para iniciantes.

Uma chave { deve ser sempre fechada por outra chave }. Essa é uma condição que é frequentemente chamada de as chaves estarem balanceadas. A IDE Arduino inclui uma forma conveniente de checar o balanço de duas chaves. Apenas escolha uma chave, ou até mesmo clique no ponto de inserção imediatamente após a chave, se a outra chave do par será destacada.

Chaves desbalanceadas podem frequentemente resultar em erros enigmáticos, que podem as vezes ser difíceis de se encontrar em um programa longo. Por causa de seu uso variado, as chaves são incrivelmente importantes para o programa e mover uma chave pode afetar dramaticamente o funcionamento de um programa.

Código de Exemplo

Os usos principais das chaves são listados nos exemplos abaixo.

Funções

void minhafuncao(tipo argumento) {

// comando(s)

}

Loops

while (expressão boolena) {

// comando(s)

}

do {

// comando(s)

} while (expressão boolena);

for (inicialização; condição; incremento) {

// comando(s)

}

Estruturas Condicionais

if (expressão boolena) {

// comando(s)

}

else if (expressão boolena) {

// comando(s)

}

else {

// comando(s)

}

Operadores Aritméticos

%

[Operadores Aritméticos]

Descrição

A operação resto calcula o resto da divisão de um inteiro por outro. Esse operador também pode ser chamado de módulo. Apesar do nome, não confunda o operador módulo (%) da computação com o módulo da matemática ( |x| ), que na linguagem arduino é calculado através da função abs(). Esse operador é útil para manter uma variável dentro de um intervalo (ex. o tamanho de um vetor). O símbolo % (porcentagem) é usado para realizar a operação resto.

Sintaxe

resto = dividendo % divisor;

Parâmetros

resto : variável. Tipos de dados permitidos: int, float, double

dividendo : variável ou constante. Tipos de dados permitidos: int

divisor : variável ou constante diferente de zero. Tipos de dados permitidos: int

Código de Exemplo

int x = 0;

x = 7 % 5; // x agora contém 2

x = 9 % 5; // x agora contém 4

x = 5 % 5; // x agora contém 0

x = 4 % 5; // x agora contém 4

/\* Atualiza um valor do vetor a cada vez em um loop \*/

int values[10];

int i = 0;

void setup() {}

void loop()

{

values[i] = analogRead(0);

i = (i + 1) % 10; // o operador módulo faz a variável i "rolar" pro próximo valor

}

Notas e Advertências

O operador resto não funciona em floats.

\*

[Operadores Aritméticos]

Descrição

Multiplicação é uma das quatro operações aritméticas fundamentais. O operador \* (asterisco) opera em dois operandos para produzir o produto.

Sintaxe

produto = operando1 \* operando2;

Parâmetros

produto : variável. Tipos de dados permitidos: int, float, double, byte, short, long

operando1 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

operando2 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

int a = 5;

int b = 10;

int c = 0;

c = a \* b; // A variável 'c' recebe o valor 50 depois que essa operação é executada

Notas e Advertências

O operação de multiplicação pode causar overflow se o resultado é maior que o que pode ser armazenado no tipo de dado.

Se um dos números (operandos) é do tipo float ou double, a matemática de ponto flutuante será usada no cálculo.

Se os operandos são do tipo de dado float ou double e a variável que armazena o produto é int, então apenas a parte inteira é salva no resultado e a parte racional é perdida.

float a = 5.5;

float b = 6.6;

int c = 0;

c = a \* b; // a variável 'c' armazena o valor 36 em vez do resultado esperado 36.3

+

[Operadores Aritméticos]

Descrição

Adição é uma das quatro operações ariméticas fundamentais. O operador + (mais) opera em dois operandos para produzir a soma dos mesmos.

Sintaxe

soma = operando1 + operando2;

Parâmetros

soma : variável. Tipos de dados permitidos: int, float, double, byte, short, long

operando1 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

operando2 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

int a = 5;

int b = 10;

int c = 0;

c = a + b; // A variável 'c' recebe o valor 15 depois que essa operação é executada

Notas e Advertências

O operação de adição pode causar overflow se o resultado é maior que o que pode ser armazenado no tipo de dado (ex. adicionar 1 a um int com valor 32,767 resulta em -32,768).

Se um dos números (operandos) é do tipo float ou double, a matemática de ponto flutuante será usada no cálculo.

Se os operandos são do tipo de dado float ou double e a variável que armazena a soma é int, então apenas a parte inteira é salva no resultado e e a parte racional é perdida.

float a = 5.5;

float b = 6.6;

int c = 0;

c = a + b; // a variável 'c' guarda o valor 12 apenas, em vez de 12.1

-

[Operadores Aritméticos]

Descrição

Subtração é uma das quatro operações aritméticas fundamentais. O operador - (menos) opera em dois operanddos para produzir a diferença do segundo pelo primeiro.

Sintaxe

diferença = operando1 - operando2;

Parâmetros

diferença : variável. Tipos de dados permitidos: int, float, double, byte, short, long

operando1 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

operando2 : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

int a = 5;

int b = 10;

int c = 0;

c = a - b; // A variável 'c' recebe o valor -5 depois que essa operação é executada

Notas e Advertências

O operação de subtração pode causar overflow se o resultado é menor que o que pode ser armazenado no tipo de dado (ex. subtrair 1 de um int com valor -32,768 resulta em 32,767).

Se um dos números (operandos) é do tipo float ou double, a matemática de ponto flutuante será usada no cálculo.

Se os operandos são do tipo de dado float ou double e a variável que armazena a diferença é int, então apenas a parte inteira é salva no resultado e a parte racional é perdida.

float a = 5.5;

float b = 6.6;

int c = 0;

c = a - b; // a variável 'c' armazena o valor -1 em vez do resultado esperado -1.1

/

[Operadores Aritméticos]

Descrição

Divisão é uma das quatro operações aritméticas fundamentais. O operador / (barra) opera em dois operandos para produzir o resultado.

Sintaxe

resultado = numerador / denominador;

Parâmetros

resultado : variável. Tipos de dados permitidos: int, float, double, byte, short, long

numerador : variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

denominador : variável ou constante diferente de zero. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

int a = 50;

int b = 10;

int c = 0;

c = a / b; // A variável 'c' recebe o valor 5 depois que essa operação é executada

Notas e Advertências

Se um dos números (operandos) é do tipo float ou double, a matemática de ponto flutuante será usada no cálculo.

Se os operandos são do tipo de dado float ou double e a variável que armazena a divisão é int, então apenas a parte inteira é salva no resultado e a parte racional é perdida.

float a = 55.5;

float b = 6.6;

int c = 0;

c = a / b; // a variável 'c' armazena o valor 8 em vez do resultado esperado 8.409

=

[Operadores Aritméticos]

Descrição

O sinal de igual = na linguagem de programação C++ é chamado operador de atribuição. Seu significado é diferente das aulas de matemática, onde ele indica uma equação ou igualdade. O operador de atribuição indica ao microcontrolador que compute qualquer valor ou expressão que estiver à direita do sinal de igual, e o armazene na variável à esquerda do sinal de igual.

Código de Exemplo

int sensVal; // declara uma variável inteira chamada sensVal

sensVal = analogRead(0); // armazena a tensão (digitalizada) do pino analógico 0 em SensVal

Notas e Advertências

A variável à esquerda do operador de atribuição ( sinal = ) precisa ser um tipo de dado capaz de armazenar o resultado da operação. Se ela não for grande o suficiente para guardar o valor, o resultado armazenado será incorreto. Veja os tipos de dados da linguagem Arduino para mais informações sobre seu tamanho.

Não confunda o operador de atribuição [ = ] (sinal de igual único) com o operador de comparação [ == ] (sinal de igual duplo), o qual avalia se se duas expressões são iguais.

Operadores de Comparação

!=

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando os operandos não são iguais. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x != y; // é falso se x é igual a y e é verdadeiro se x não é igual a y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x != y) { // testa se x é diferente de y

// faz algo apenas se o resultado da comparação é verdadeiro

}

<

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando o operando à esquerda é menor que o operando à direita. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x < y; // é verdadeiro se x é menor que y e é falso se x é igual ou maior que y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x < y) { // testa se x é menor que y

// faz algo apenas se o resultado da comparação é verdadeiro

}

Notas e Advertências

Números positivos são maiores que números negativos.

<=

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando o operando à esquerda é menor ou igual ao operando à direita. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x <= y; // é verdadeiro se x é menor ou igual a y e é falso se x é maior que y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x <= y) { // testa se x é menor que ou igual a y

// faz algo apenas se o resultado da comparação é verdadeiro

}

Notas e Advertências

Números positivos são maiores que números negativos.

==

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando os dois operandos são iguais. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x == y; // é verdadeiro se x é igual a y e é falso se x não é igual a y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x == y) { // testa se x é igual a y

// faz algo apenas se o resultado da comparação é verdadeiro

}

>

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando o operando à esquerda é maior que o operando à direita. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x > y; // é verdadeiro se x é maior que y e é falso se x é igual ou menor que y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x > y) { // testa se x é maior que y

// faz algo apenas se o resultado da comparação é verdadeiro

}

==

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando os dois operandos são iguais. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x == y; // é verdadeiro se x é igual a y e é falso se x não é igual a y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x == y) { // testa se x é igual a y

// faz algo apenas se o resultado da comparação é verdadeiro

}

>=

[Operadores de Comparação]

Descrição

Compara a variável à esquerda com o valor ou variável à direita do operador. Retorna verdadeiro (true) quando o operando à esquerda é maior ou igual ao operando à direita. Por favor note que você pode comparar variáveis de tipos de dados diferentes, mas isso pode gerar resultados imprevisíveis. Por isso é recomendado comparar variáveis do mesmo tipo de dados incluindo o atributo signed/unsigned.

Sintaxe

x >= y; // é verdadeiro se x é maior ou igual a y e é falso se x é menor que y

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

if (x >= y) { // testa se x é maior que ou igual a y

// faz algo apenas se o resultado da comparação é verdadeiro

}

Operadores Boleanos

!

[Operadores Boleanos]

Descrição

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 true

// código a ser executado

}

Pode ser usado para inverter o valor booleano.

x = !y; // o valor booleano invertido de y é guardado em x

Notas e Advertências

O NÃO bitwise ~ (til) é muito diferente do NÃO booleano ! (ponto de exclamação), mas você ainda deve ter certeza de qual você precisa usar.

&&

[Operadores Boleanos]

Descrição

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 verdadeiras

}

Notas e Advertências

Tome cuidado para não confundir o operatdor booleano E, && (e comercial duplo); com operador bitwise E, & (e comercial único). Eles são usados para propósitos diferentes.

||

[Operadores Boleanos]

Descrição

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

}

Notas e Advertências

Não confunda o operador OU booleano || (pipe duplo) com o operador OU bitwise | (pipe único).

|

[Operadores Bitwise]

Descrição

O operador bitwise OU em C++ é o caractere barra vertical, |. Da mesma forma que o operador &, | opera independentemente em cada bit dos dois operandos, mas seu propósito é diferente (duh!). O resultado da operação OU entre dois bits é 1 se qualquer um ou ambos os bits de entrada são 1, do contrário é 0.

Explicado de outra forma:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 1 1 1 (operando1 | operando2) - resultado retornado

Código de Exemplo

int a = 92; // em binário: 0000000001011100

int b = 101; // em binário: 0000000001100101

int c = a | b; // resultado: 0000000001111101, ou 125 em decimal.

Um dos usos mais comuns do OU bit-a-bit é "setar" um ou mais bits em um valor.

// Nota: Esse código é específico para a arquitetura AVR

// configura os bits de direção para os pinos 2 a 7, deixa 0 e 1 intocados (xx | 00 == xx)

// Equivale a pinMode(pino, OUTPUT) para os pinos 2 a 7 no Arduino Uno ou Nano

DDRD = DDRD | B11111100;

// o mesmo que pinMode(pino, OUTPUT) para os pinos 2 a 7

Operadores para Ponteiros

&

[Operadores para Ponteiros]

Descrição

Referência é uma das características da linguagem C para uso especificamente com ponteiros. O operador & ('e' comercial ou ampersand) é utilizado para esse propósito. Se x é uma variável, então &x representa o endereço da variável x.

Código de Exemplo

int \*p; // declara um ponteiro para uma variável do tipo int

int i = 5, resultado = 0;

p = &i; // agora 'p' contém o endereço de 'i'

resultado = \*p; // 'resultado' recebe o valor contido no endereço apontado por 'p'

// isto é, recebe o valor de 'i', que é 5

Notas e Advertências

Ponteiros são um dos assuntos mais complicados para iniciantes na linguagem C aprenderem, e é possível escrever a grande maioria dos sketches Arduino sem nunca encontrar ponteiros. No entanto, para manipular certas estruturas de dados, o uso de ponteiros pode simplificar o código. O conhecimento da manipulação de ponteiros é vantajoso de se ter no seu kit de ferramentas.

\*

[Operadores para Ponteiros]

Descrição

Desreferência é uma das características da linguagem C para uso especificamente com ponteiros. O operador \* (asterisco) é utilizado para esse propósito. Se p é um ponteiro, então \*p representa o valor contido no endereço apontado por p.

Código de Exemplo

int \*p; // declara um ponteiro para uma variável do tipo int

int i = 5;

int resultado = 0;

p = &i; // agora 'p' contém o endereço de 'i'

resultado = \*p; // 'resultado' recebe o valor contido no endereço apontado por 'p'

// isto é, recebe o valor de 'i', que é 5

Notas e Advertências

Ponteiros são um dos assuntos mais complicados para iniciantes na linguagem C aprenderem, e é possível escrever a grande maioria dos sketches Arduino sem nunca encontrar ponteiros. No entanto, para manipular certas estruturas de dados, o uso de ponteiros pode simplificar o código. O conhecimento da manipulação de ponteiros é vantajoso de se ter no seu kit de ferramentas.

Operadores Bitwise

&

[Operadores Bitwise]

Descrição

O operador bitwise E em C++ é um único & (e comercial ou ampersand), usado entre duas expressões inteiras. O operador E bit-a-bit atua em cada bit dos operandos independentemente, de acordo com a seguinte regra: se ambas entradas são 1, a saída resultante é 1, do contrário o resultado é 0.

Outra forma de se expressão isso é:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 0 0 1 (operando1 & operando2) - resultado retornado

No Arduino, o tipo int é um valor 16-bit, então & usado entre dois operandos do tipo int causa 16 operações bit-a-bit E simultâneas.

Código de Exemplo

Em um fragmento de código do tipo:

int a = 92; // em binário: 0000000001011100

int b = 101; // em binário: 0000000001100101

int c = a & b; // resultado: 0000000001000100, ou 68 em decimal.

Cada um dos 16 bits em a e b são processados com o operador &, e todos os 16 bits resultantes são armazenados em c, resultando no valor binário 01000100, que é 68 em decimal.

Um dos usos mais comuns da operação bit-a-bit E é selecionar um bit particular (ou bits) de um valor inteiro, frequentemente chamado de masking. Veja abaixo um exemplo (específico para arquitetura AVR):

PORTD = PORTD & B00000011; // apaga os bits 2 - 7, deixando o valor dos pinos PD0 e PD1 intocados (xx & 11 == xx)

\*=

[Operadores de Atribuição Composta]

Descrição

Essa é uma abreviação conveniente para realizar a multiplicação de uma variável com outra variável ou constante.

Sintaxe

x \*= y; // equivalente a expressão x = x \* y;

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

x = 2;

x \*= 2; // x agora contém 4

&=

[Operadores de Atribuição Composta]

Descrição

O operador de atribuição composta E &= é frequentemente usado com uma variável e uma constante para forçar detreminados bits de uma variável para 0. Isso é frequentemente chamado de "limpar" ou "resetar" os bits.

Uma revisão do operator bitwise E &:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 0 0 1 (operando1 & operando2) - resultado retornado

Sintaxe

x &= y; // equivalente a x = x & y;

Parâmetros

x: variável. Tipos de dados permitidos: char, int, long

y: variável ou constante. Tipos de dados permitidos: char, int, long

Código de Exemplo

Bits que são operados através do E com 0 são limpos, então se meuByte é uma variável byte:,

meuByte & B00000000 = 0;

Bits que são operados através do E com 1 não são mudados, então se meuByte é uma variável byte:

meuByte & B11111111 = meuByte;

Notas e Advertências

Porque estamos lidando com bits em um operador bitwise, é conveniente usar o modificador binário com constantes na representação. Os números ainda são os mesmos em outras representações, só não são tão fáceis de entender. Também, B00000000 é mostrado para claridade, mas zero em qualquer outro formato é zero.

Consequentemente - para limpar os bits 0 e 1 de um variável, ao mesmo tempo deixando o resto da variável intocado, useo o operador de atribuição por bitwise E (&=) com a constante B11111100

1 0 1 0 1 0 1 0 variável

1 1 1 1 1 1 0 0 máscara

----------------------

1 0 1 0 1 0 0 0

bits intactos

bits limpos

Abaixo a mesma representação com os bits da variável substituidos pelo símbolo x

x x x x x x x x variável

1 1 1 1 1 1 0 0 máscara

----------------------

x x x x x x 0 0

bits intactos

bits limpos

Então:

myByte = B10101010;

myByte &= B11111100; // resulta em B10101000

++

[Operadores de Atribuição Composta]

Descrição

Incrementa o valor de uma variável em 1.

Sintaxe

x++; // incrementa x em um e retorna o valor antigo de x

++x; // incrementa x em um e retorna o novo valor de x

Parâmetros

x: variável. Tipos de dados permitidos: integer, long (possibly unsigned)

Retorna

O valor original ou incrementado da variável, que depende se o operador está à esquerda ou direita da variável.

Código de Exemplo

x = 2;

y = ++x; // x agora contém 3, y contém 3

y = x++; // x contém 4, mas y ainda contém 3

+=

[Operadores de Atribuição Composta]

Descrição

Essa é uma abreviação conveniente para realizar a adição de uma variável com outra variável ou constante.

Sintaxe

x += y; // equivalente a expressão x = x + y;

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

x = 2;

x += 4; // x agora contém 6

--

[Operadores de Atribuição Composta]

Descrição

Decrementa o valor de uma variável em 1.

Sintaxe

x--; // decrementa x em um e retorna o valor antigo de x

--x; // decrementa x em um e retorna o novo valor de x

Parâmetros

x: variável. Tipos de dados permitidos: integer, long (possibly unsigned)

Retorna

O valor original ou decrementado da variável, que depende se o operador está à esquerda ou direita da variável.

Código de Exemplo

x = 2;

y = --x; // x agora contém 1, y contém 1

y = x--; // x contém 0, mas y ainda contém 1

/=

[Operadores de Atribuição Composta]

Descrição

Essa é uma abreviação conveniente para realizar a divisão de uma variável com outra variável ou constante.

Sintaxe

x /= y; // equivalente a expressão x = x / y;

Parâmetros

x: variável. Tipos de dados permitidos: int, float, double, byte, short, long

y: variável ou constante diferente de zero. Tipos de dados permitidos: int, float, double, byte, short, long

Código de Exemplo

x = 2;

x /= 2; // x agora contém 1

^=

[Operadores de Atribuição Composta]

Descrição

O operador de atribuição composta bitwise OU EXCLUSIVO ^= é frequentemente usado com uma variável e uma constante para "inverter" (ou trocar) bits particulares de uma variável.

Uma revisão do operador bitwise OU EXCLUSIVO ^:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 1 1 0 (operando1 ^ operando2) - resultado retornado

Sintaxe

x ^= y; // equivalente a x = x ^ y;

Parâmetros

x: variável. Tipos de dados permitidos: char, int, long

y: variável ou constante. Tipos de dados permitidos: char, int, long

Código de Exemplo

Bits que são operados através do OU OEXCLUSIVO com 0 não são mudados, então se meuByte é uma variável byte:

meuByte ^ B00000000 = meuByte;

Bits que são operados através do OU EXCLUSIVO com 1 são invertidos, então:

meuByte ^ B11111111 = ~meuByte;

Notas e Advertências

Porque estamos lidando com bits em um operador bitwise, é conveniente usar o modificador binário com constantes na representação. Os números ainda são os mesmos em outras representações, só não são tão fáceis de entender. Também, B00000000 é mostrado para claridade, mas zero em qualquer outro formato é zero.

Consequentemente, para inverter os bits 0 e 1 de um variável ao mesmo tempo, deixando o resto da variável intocado, use o operador de atribuição por bitwise OU EXCLUSIVO (^=) com a constante B00000011

1 0 1 0 1 0 1 0 variável

0 0 0 0 0 0 1 1 máscara

----------------------

1 0 1 0 1 0 0 1

bits intactos

bits invertidos

Abaixo a mesma representação com os bits da variável substituidos pelo símbolo x. ~x representa o complemento de x.

x x x x x x x x variável

0 0 0 0 0 0 1 1 máscara

----------------------

x x x x x x ~x ~x

bits intactos

bits invertidos

Então:

myByte = B10101010;

myByte ^= B00000011 == B10101001;

|=

[Operadores de Atribuição Composta]

Descrição

O operador de atribuição composta bitwise OU |= é frequentemente usado com uma variável e uma constante para "setar" (mudar pra 1) bits particulares em uma variável.

Uma revisão do operador bitwise OU |:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 1 1 1 (operando1 | operando2) - resultado retornado

Sintaxe

x |= y; // equivalente a x = x | y;

Parâmetros

x: variável. Tipos de dados permitidos: char, int, long

y: variável ou constante. Tipos de dados permitidos: char, int, long

Código de Exemplo

Bits que são operados através do OU com 0 não são mudados, então se meuByte é uma variável byte:

meuByte | B00000000 = meuByte;

Bits que são operados através do OU com 1 são mudados para 1, então:

meuByte | B11111111 = B11111111;

Notas e Advertências

Porque estamos lidando com bits em um operador bitwise, é conveniente usar o modificador binário com constantes na representação. Os números ainda são os mesmos em outras representações, só não são tão fáceis de entender. Também, B00000000 é mostrado para claridade, mas zero em qualquer outro formato é zero.

Consequentemente - para setar os bits 0 e 1 de um variável, ao mesmo tempo deixando o resto da variável intocado, useo o operador de atribuição por bitwise OU (|=) com a constante B00000011

1 0 1 0 1 0 1 0 variável

0 0 0 0 0 0 1 1 máscara

----------------------

1 0 1 0 1 0 1 1

bits intactos

bits mudados para 1

Abaixo a mesma representação com os bits da variável substituidos pelo símbolo x

x x x x x x x x variável

0 0 0 0 0 0 1 1 máscara

----------------------

x x x x x x 1 1

bits intactos

bits mudados para 1

Então:

meuByte = B10101010;

meuByte |= B00000011; // igual a B10101011;

|

[Operadores Bitwise]

Descrição

O operador bitwise OU em C++ é o caractere barra vertical, |. Da mesma forma que o operador &, | opera independentemente em cada bit dos dois operandos, mas seu propósito é diferente (duh!). O resultado da operação OU entre dois bits é 1 se qualquer um ou ambos os bits de entrada são 1, do contrário é 0.

Explicado de outra forma:

0 0 1 1 operando1

0 1 0 1 operando2

----------

0 1 1 1 (operando1 | operando2) - resultado retornado

Código de Exemplo

int a = 92; // em binário: 0000000001011100

int b = 101; // em binário: 0000000001100101

int c = a | b; // resultado: 0000000001111101, ou 125 em decimal.

Um dos usos mais comuns do OU bit-a-bit é "setar" um ou mais bits em um valor.

// Nota: Esse código é específico para a arquitetura AVR

// configura os bits de direção para os pinos 2 a 7, deixa 0 e 1 intocados (xx | 00 == xx)

// Equivale a pinMode(pino, OUTPUT) para os pinos 2 a 7 no Arduino Uno ou Nano

DDRD = DDRD | B11111100;

// o mesmo que pinMode(pino, OUTPUT) para os pinos 2 a 7