LINGUAGUEM ARDUINO

ESTRUTURA

#include

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).

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
  }
}
```

#define

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
```

/* */

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 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.

//

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.

;

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.

{ }

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, e 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)
}
```