Programação em C++

- Funções e variáveis
- Comunicação serial

Funções e variáveis

Agora que foi apresentado sobre as interfaces de programação que são utilizadas para configurar e simular microcontroladores, mostraremos nesta seção como funciona a linguagem C++ e o básico para que seja possível compreender os códigos das próximas seções e como criar os seus próprios.

Funções Básicas

No ambiente de programação do Arduino, as funções principais são `setup` e `loop`. Ambas têm papéis específicos na estrutura do programa e são essenciais para o funcionamento do código.

A função setup prepara o ambiente de execução. A função loop mantém o microcontrolador em operação contínua, executando tarefas de forma cíclica. Essa separação simplifica o desenvolvimento, organizando o código de forma clara e eficiente.

Função Setup

A função setup é executada apenas uma vez, logo após o microcontrolador ser ligado ou reiniciado. Ela é usada para configurar o ambiente inicial do programa, como:

- Configurar pinos como entrada ou saída.
- Inicializar comunicações seriais (ex.: Serial.begin(9600);).
- Configurar periféricos e bibliotecas necessárias para o funcionamento do programa.

Exemplo:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Define o LED como saída
}
```

Função Loop

A função loop é executada continuamente após a conclusão da setup. Todo o código que define o comportamento repetitivo ou contínuo do microcontrolador deve ser implementado aqui.

Exemplo:

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // Liga o LED
  delay(1000);                     // Aguarda 1 segundo
  digitalWrite(LED_BUILTIN, LOW);  // Desliga o LED
  delay(1000);                     // Aguarda 1 segundo
}
```

Tipos de variáveis em C++ e seus principais usos

Ao programar em C++ para aplicações de microcontroladores, é essencial entender os diferentes tipos de variáveis disponíveis. Cada tipo de variável é projetado para armazenar dados de formas específicas, otimizando o uso de memória e garantindo a eficiência do código. Abaixo, explicamos os principais tipos de variáveis em C++ e seus usos mais comuns:

Variáveis Inteiras

Variáveis inteiras são usadas para armazenar números inteiros (positivos, negativos ou zero), sem casas decimais.

- **Tipo "int":** É o tipo inteiro mais utilizado. Armazena valores entre -32.768 e 32.767 (em sistemas de 16 bits) ou entre -2.147.483.648 e 2.147.483.647 (em sistemas de 32 bits).
  Uso: Contadores, índices de laços, medições que não requerem frações.
- **Tipo "short":** Um tipo inteiro menor, que ocupa menos memória.
  Uso: Ideal para situações onde a memória é limitada e o intervalo de valores é pequeno.
- **Tipo "long" e "long long":** Usados para armazenar números inteiros muito grandes.
  Uso: Cálculos que exigem maior precisão ou intervalos mais amplos.
- **Modificador "unsigned":** Indica que a variável não armazenará valores negativos, dobrando o limite superior.
  Uso: Contagem de eventos ou medições sempre positivas, como leituras de sensores.

Variáveis de Ponto Flutuante

Usadas para representar números reais (com casas decimais).

- **Tipo "float":** Representa números com precisão simples (32 bits).
  Uso: Cálculos que exigem precisão moderada, como medições analógicas ou

coordenadas em um sistema.
- **Tipo "double":** Representa números com precisão dupla (64 bits).
  Uso: Cálculos mais complexos que exigem maior precisão.

Nota: Em microcontroladores com recursos limitados, é importante usar `float` ou `double` apenas quando necessário, pois essas variáveis consomem mais memória e poder de processamento.

Variáveis de Caractere

- **Tipo "char":** Usado para armazenar um único caractere (como 'A', '1', ou '?') ou valores numéricos pequenos (de 0 a 255 no caso de `unsigned char`).
  Uso: Representa caracteres individuais ou pequenos números inteiros, como códigos ASCII.

Variáveis Booleanas

- **Tipo "bool":** Armazena valores lógicos, ou seja, `true` (verdadeiro) ou `false` (falso).
  Uso: Controle de fluxos lógicos, como verificar se uma condição foi atendida.

Variáveis Compostas

- **Tipo "array":** Conjunto de variáveis do mesmo tipo armazenadas em sequência.
  Uso: Armazenar múltiplos valores relacionados, como leituras de um sensor ao longo do tempo.
- **Estruturas (struct):** Agrupam variáveis de diferentes tipos sob um único nome.
  Uso: Representação de objetos mais complexos, como as informações de um dispositivo.

Ponteiros

- *Tipo* "*int*", "*float*", etc.:** Variáveis que armazenam o endereço de memória de outra variável.
  Uso: Manipulação direta de memória, comunicação com periféricos ou passagem eficiente de dados entre funções.

Considerações Importantes

Ao trabalhar com microcontroladores, é essencial escolher o tipo de variável adequado para:

- **Otimizar o uso de memória:** Microcontroladores geralmente têm recursos limitados.
- **Garantir eficiência:** Variáveis menores permitem operações mais rápidas.

- **Evitar erros:** Certifique-se de que o intervalo de valores suportado pela variável é suficiente para a aplicação.

Comunicação serial

Comunicação serial

A comunicação serial é um dos meios mais simples e eficientes para enviar e receber dados entre o microcontrolador e outros dispositivos, como computadores ou módulos externos. No Arduino, a biblioteca `Serial` facilita esse tipo de comunicação utilizando o protocolo UART.

Como Funciona

- **Velocidade de Comunicação:** Definida em bauds (ex.: 9600, 115200), representa o número de bits transferidos por segundo.
- **Transmissão e Recepção:** Usa dois pinos — TX (transmissão) e RX (recepção) ou comunicação direta com o computador na IDE por cabo.
- **Formato de Dados:** Tipicamente, cada pacote de dados inclui 8 bits, 1 bit de parada e nenhum bit de paridade.

Configuração inicial

Antes de utilizar a comunicação serial, é necessário inicializá-la na função `setup` com o comando:

```
Serial.begin(9600);
```

Envio e recebimento de dados

Para enviar dados do microcontrolador, utiliza-se o comando:

```
Serial.print("Mensagem"); // Envia uma mensagem sem pular linha
Serial.println("Mensagem"); // Envia uma mensagem e adiciona uma nova linha
```

Para receber dados, é possível verificar se há informações disponíveis com:

```
if (Serial.available() > 0) {
  char dado = Serial.read(); // Lê um byte disponível
```

```
  Serial.println(dado); // Ecoa o dado recebido
}
```

Exemplo completo

```
void setup() {
    Serial.begin(9600); // Inicializa a comunicação serial
    Serial.println("Iniciando comunicação serial...");
}

void loop() {
  if (Serial.available() > 0) {
    char recebido = Serial.read();
    Serial.print("Você enviou: ");
    Serial.println(recebido);
  }
  delay(100); // Pequena pausa para evitar sobrecarga
}
```

Aplicações comuns

- Depuração e monitoramento de código.
- Comunicação com módulos Bluetooth, Wi-Fi, ou GPS.
- Envio de comandos para o microcontrolador a partir de um computador ou outro dispositivo.

Compreender a comunicação serial é essencial para criar projetos interativos e para monitorar o comportamento do microcontrolador durante a execução de um programa.