Revisão da lógica de programação em C para aplicações em microcontroladores

Tipos de Dados Básicos

Os tipos de dados em C são fundamentais para definir como o compilador irá interpretar, armazenar e manipular informações dentro da memória do microcontrolador.

- char: ocupa 1 byte, pode armazenar caracteres ou números pequenos.
- unsigned char: 0 a 255. Muito utilizado para manipulação de registradores de 8 bits.
- signed char: -128 a 127.
- int: 1, 2 ou 4 bytes, depende da arquitetura do microcontrolador.
- unsigned int: apenas valores positivos, aumentando o limite superior.
- short int e long int: modificadores para garantir tamanhos específicos.
- float: 4 bytes. Permite trabalhar com números decimais. Deve ser usado com cautela, pois operações de ponto flutuante são lentas em MCUs.
- double: 8 bytes. Alta precisão, porém consome muita memória e processamento.

Tipos Especiais de Dados

Além dos tipos básicos, C permite organizar melhor as informações com estruturas e enumerações:

- struct: agrupa variáveis de tipos diferentes em um único bloco. Ideal para representar dados de sensores. Permite acessar dados de forma organizada.
- enum: cria um conjunto de constantes inteiras nomeadas. Torna o código mais legível e facilita a implementação de máquinas de estados.
- Exemplo:

```
enum EstadoMotor { PARADO, FRENTE, REVERSO };
```

```
struct SensorData {
int id;
float temperatura;
unsigned char umidade;
 };
struct SensorData sensor_sala;
sensor_sala.temperatura = 25.7;
```

Operadores Aritméticos e Lógicos

- Soma (+), Subtração (-), Multiplicação (*), Divisão (/), Módulo (%).
- O operador % retorna o resto da divisão, útil em contadores cíclicos.
  - Exemplo: 10 % 3 resulta em 1.

Operadores lógicos permitem tomar decisões:

- && (E lógico): verdadeiro apenas se as duas condições forem verdadeiras.
- || (OU lógico): verdadeiro se pelo menos uma condição for verdadeira.
- ! (NÃO lógico): inverte o valor lógico.

Operadores de Bit (Bitwise) e Bit Shift

Essenciais em programação de microcontroladores, pois permitem manipular bits individuais:

- & (AND): usado para limpar um bit específico.
- | (OR): usado para ativar um bit.
- ^ (XOR): inverte um bit.
- ~ (NOT): inverte todos os bits.

Deslocamento de bits:

- << (shift para esquerda): multiplica por 2^n.
- >> (shift para direita): divide por 2^n.
    - Exemplo prático: PORTB = PORTB | (1 << 5); ativa o bit 5 do registrador PORTB.

Cast de Dados (Conversão de Tipos)

O cast força uma variável a ser interpretada como outro tipo. É útil para evitar erros em cálculos.

- Exemplo:

```
int a = 5,
b = 2;
float resultado;
resultado = a / b; // resulta em 2.0 (errado)
resultado = (float)a / b; // resulta em 2.5 (correto).
```

- Também é importante ao passar valores para funções que esperam tipos específicos.

Estrutura if, else e else if

Estrutura If

O bloco if executa código quando a condição é verdadeira, iniciando processos condicionais.

Estrutura Else

Else define o caminho alternativo quando a condição if não é satisfeita, garantindo fluxo lógico.

Estrutura Else If

Else if permite múltiplas condições sequenciais, facilitando decisões complexas e ramificadas.

```
if (temperatura > 40.0)
      {
         liga_ventilador();
      }
else if (temperatura < 10.0)
      {
         liga_aquecedor();
      }
else
      {
         desliga_tudo();
      }
```

Estrutura switch case

Avaliação da Variável

A estrutura 'switch' avalia uma variável para determinar qual bloco de código executar.

Uso do 'case'

O 'case' define valores específicos que, quando coincidem, executam blocos de código correspondentes.

Alternativa ao if else

Substitui múltiplos 'if else' para condições de igualdade, melhorando a legibilidade do código.

```
switch (tecla_pressionada) {
    case '1':
        ativa_funcao_1();
        break; // O 'break' é
fundamental!
    case '2':
        ativa_funcao_2();
        break;
    default:
        mostra_erro();
        break;
}
```

Estrutura for: inicialização, condição e incremento

Controle do Laço For

O laço 'for' organiza inicialização, condição e incremento para controlar repetições de forma clara e compacta.

Uso em Loops Definidos

É ideal para loops com número fixo de iterações, como percorrer vetores ou listas em programação.

Execução de Tarefas Repetitivas

Permite executar tarefas repetitivas limitadas com controle preciso do número de repetições.

```
// Pisca um LED 10 vezes
for (int i = 0; i < 10; i++)
        {
        liga_led();
        delay(100);
        desliga_led();
        delay(100);
        }
```

Estrutura *while*: teste antes da execução

Teste da Condição Antes do Loop

A condição do while é avaliada antes do bloco executar, garantindo execução condicional segura e controlada.

Execução Condicional do Bloco

O código dentro do while roda somente se a condição for verdadeira, evitando execuções desnecessárias.

Uso com Variáveis Mutáveis

O while é útil para loops que dependem de estados externos ou variáveis que mudam durante a execução.

```
// Espera um botão ser pressionado
while (le_botao() == SOLTO)
        {
        // Não faz nada, apenas espera
        }
```

Estrutura *do while*: execução garantida ao menos uma vez

Execução Inicial Garantida

O 'do while' sempre executa o código uma vez antes de verificar a condição.

Diferença para o While

Ao contrário do 'while', o 'do while' não testa a condição antes da primeira execução.

Aplicação Prática

Útil quando uma ação deve ocorrer pelo menos uma vez independentemente da condição inicial.

```
int temperatura;
do {
temperatura = ler_sensor_temperatura();
delay(50);
 } while (temperatura < 100);
// A condição só é testada no final
```

Declaração e inicialização de arrays

Declaração de arrays

Arrays são declarados especificando o tipo de dado e o tamanho fixo para armazenar múltiplos valores sequenciais.

Inicialização de arrays

A inicialização pode ser feita durante a declaração ou posteriormente para preencher o array com valores organizados.

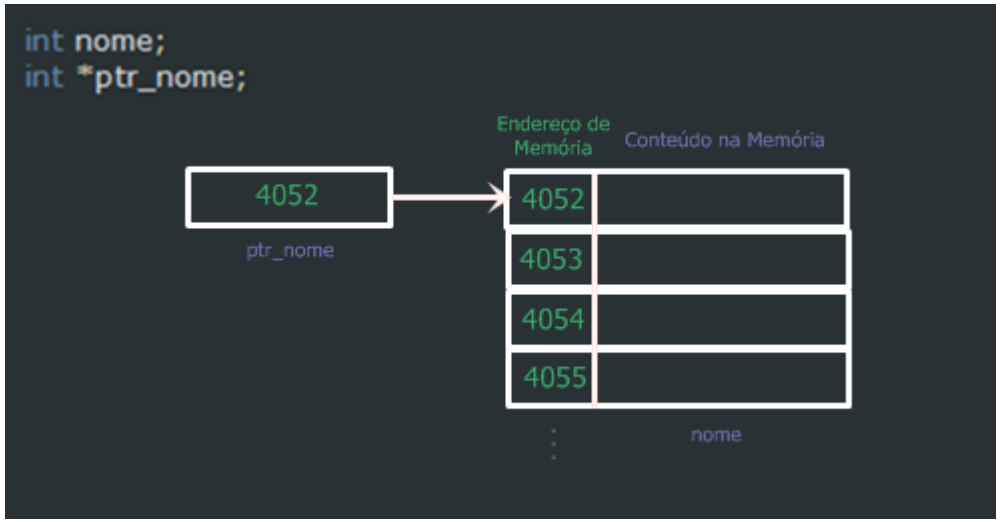
Aplicações comuns:

- armazenar leituras de sensores (ex.: ADC).
- criar buffers para comunicação serial (UART).
- lookup tables para cálculos.
    - Exemplo: float leituras[50];

Declaração e inicialização de ponteiros

O que são ponteiros?

- Um ponteiro é uma variável especial que não guarda um valor diretamente, mas sim o endereço de memória onde esse valor está armazenado.
- Analogia: Imagine a memória do microcontrolador como uma enorme prateleira com milhares de gavetas numeradas (endereços).
- Uma variável normal **(int x = 10;)** guarda o valor dentro de uma gaveta. Um ponteiro guarda o número da gaveta onde a variável está. Assim, se x está na **gaveta número 1000**, um ponteiro *px* guardaria 1000 em vez de 10.



Para declarar um ponteiro em C usamos o operador *.

Exemplo:

```
int valor = 10;     // variável comum
int *ptr;           // ponteiro para inteiro
ptr = &valor;       // ponteiro recebe o endereço de valor
```

Explicação passo a passo:

int *ptr; → cria um ponteiro que só pode apontar para variáveis do tipo int.

&valor; → operador & retorna o endereço da variável valor.

ptr = &valor; → agora ptr contém o endereço da variável valor.

Se quisermos acessar o valor guardado nesse endereço, usamos o operador * :

```
printf("%d", *ptr);  // imprime 10
```

Ou seja:

ptr → endereço da variável (ex.: 1000).

*ptr → valor armazenado nesse endereço (ex.: 10).