# Módulo 10 - Funções

Funções são blocos de código que só rodam se forem chamados através de seu nome. C é uma linguagem estruturada em blocos, isso significa que o compilador secciona e esconde algumas partes do programa. Isso é conseguido através das chaves, onde é possível declarar variáveis que existem somente dentro deste trecho de código, tornando-o isolado do resto do programa. A vantagem é que estes trechos isolados podem ser reutilizados em outras partes do código diversas vezes, através das funções.

Veja o exemplo do seguinte trecho de código:

```
#include <stdio.h>

void imprimir(a, b, c){
    printf("%d %d %d\n", a, b, c);
}

int main(void){
    imprimir(1, 2, 3);
}
```

```
#include <stdio.h>

// Função soma
void soma(int a, int b){
    int resultado;
    resultado = a + b;
    printf("O resultado é: %d\n", resultado);
}

// Função principal
int main(void){
    soma(2, 2); // Soma 2 + 2
    soma(2, 3); // Soma 2 + 3
    soma(5, 5); // Soma 5 + 5

    return 0;
}
```

Sobre funções:

- `printf` e `scanf` são funções da biblioteca `#include <stdio.h>` (standard input output) de C

- Uma função pode ter um especificador de tipo, que se não for declarado é int por padrão, o nome da função (que segue as regras dos identificadores).
- As funções podem ter nomes iguais se tiverem argumentos diferentes
- A compilação é feita de cima para baixo, então se as funções não estiverem declaradas antes de serem chamadas, vai ocorrer um erro.

## Escopo de variáveis

Variáveis em programação podem ser classificadas como locais ou globais, dependendo de seu escopo e de onde elas são declaradas. Aqui estão as principais diferenças entre variáveis locais e globais:

**Variáveis Locais:**

1. **Escopo**: Variáveis locais são declaradas dentro de uma função ou bloco específico e só podem ser acessadas dentro desse escopo. Elas não são visíveis fora da função ou bloco onde foram definidas.
2. **Tempo de Vida**: A vida útil de uma variável local está limitada ao tempo em que a função ou bloco em que ela está declarada está ativa. Ela é criada quando a função ou bloco é executado e destruída quando a função ou bloco é concluído.
3. **Acesso**: Variáveis locais não podem ser acessadas de fora da função ou bloco em que são declaradas. Elas são usadas para armazenar valores temporários e intermediários em uma função.
4. **Inicialização**: As variáveis locais não são inicializadas automaticamente. Seus valores iniciais são indefinidos a menos que você atribua um valor a elas explicitamente.

Exemplo de variável local:

```
#include <stdio.h>

void minhaFuncao() {
    int x = 10; // Variável local
    printf("Valor de x: %d\n", x);
}

int main() {
    minhaFuncao();
    // x não é visível aqui
    return 0;
}
```

**Variáveis Globais:**

1. **Escopo**: Variáveis globais são declaradas fora de qualquer função e, portanto, são acessíveis a partir de qualquer parte do programa, incluindo funções.
2. **Tempo de Vida**: Variáveis globais existem durante toda a execução do programa. Elas são criadas quando o programa começa a ser executado e destruídas quando o programa termina.
3. **Acesso**: Variáveis globais podem ser acessadas e modificadas em qualquer função dentro do programa. Elas são usadas para armazenar valores que devem ser compartilhados entre diferentes partes do código.
4. **Inicialização**: Variáveis globais são inicializadas automaticamente com zero ou um valor padrão, dependendo do tipo de dado.

Exemplo de variável global:

```
#include <stdio.h>

int globalVar = 20; // Variável global

void minhaFuncao() {
    printf("Valor de globalVar: %d\n", globalVar);
}

int main() {
    minhaFuncao();
    printf("Valor de globalVar: %d\n", globalVar);
    return 0;
}
```

No exemplo acima, `globalVar` é uma variável global e pode ser acessada tanto na função `minhaFuncao` quanto na função `main`.

É importante usar variáveis locais quando você precisa armazenar valores temporários dentro de funções e variáveis globais quando você precisa compartilhar dados entre diferentes partes do programa. O uso adequado de variáveis locais e globais ajuda a melhorar a organização e a modularidade do código. No entanto, é importante ter cuidado ao usar variáveis globais, pois elas podem tornar o código menos legível e mais difícil de depurar se usadas indiscriminadamente.

## Chamada por valor e chamada por referência

A diferença entre chamada por valor e chamada por referência está relacionada a como os parâmetros de uma função são passados e manipulados. Isso tem

implicações significativas no comportamento das funções e na forma como as variáveis são afetadas. Aqui está a explicação das duas abordagens:

**Chamada por Valor:**

1. **Passagem de Parâmetros**: Na chamada por valor, os valores das variáveis originais são passados como argumentos para a função. Isso significa que uma cópia dos valores é criada e usada dentro da função.
2. **Efeito nas Variáveis Originais**: As operações realizadas nos parâmetros dentro da função não afetam as variáveis originais fora da função. Qualquer alteração feita nos parâmetros dentro da função não é refletida nas variáveis originais.
3. **Utilização**: A chamada por valor é usada quando se deseja preservar os valores originais das variáveis fora da função e evitar que essas variáveis sejam modificadas pela função.

Exemplo de chamada por valor:

```
void dobro(int x) {
    x = x * 2;
}

int main() {
    int num = 5;
    dobro(num);
    // 'num' ainda é 5, não foi afetado pela função 'dobro'
    return 0;
}
```

**Chamada por Referência:**

1. **Passagem de Parâmetros**: Na chamada por referência, os endereços de memória das variáveis originais são passados como argumentos para a função. Isso permite que a função acesse e modifique diretamente as variáveis originais usando esses endereços.
2. **Efeito nas Variáveis Originais**: As operações realizadas nos parâmetros dentro da função afetam as variáveis originais fora da função. Qualquer alteração feita nos parâmetros dentro da função é refletida nas variáveis originais.
3. **Utilização**: A chamada por referência é usada quando se deseja que uma função modifique diretamente as variáveis fora dela ou quando se deseja evitar a sobrecarga de copiar grandes quantidades de dados na memória.

Exemplo de chamada por referência:

```
void dobroPorReferencia(int *x) {
    *x = *x * 2;
}

int main() {
    int num = 5;
    dobroPorReferencia(&num);
    // 'num' agora é 10, foi modificado pela função 'dobroPorReferencia'
    return 0;
}
```

Em resumo, a principal diferença entre chamada por valor e chamada por referência está na forma como as variáveis originais são afetadas por uma função. A chamada por valor cria cópias dos valores originais, enquanto a chamada por referência permite que a função acesse e modifique diretamente as variáveis originais usando ponteiros ou referências. A escolha entre as duas abordagens depende dos requisitos específicos de um programa e do comportamento desejado da função em relação às variáveis passadas como argumentos.

## Como fazer uma função retornar um array

```
// Recebe um endereço, uma quantidade de caracteres a ser trazidos e um array onde eles
// serão armazenados
void EEPROM_Read_array(unsigned char adr, unsigned char qtd_byte, unsigned char *arr){
    for (char i=0; i<qtd_byte; i++){
        *arr = EEPROM_Read_byte(adr+i);
        arr++;
    }
}
```

A função acima deve retornar o array presente na sequência de caracteres presente na memória da eeprom. Ela faz um for partindo do endereço passado até a quantidade total de valores requeridos. A cada incremento ela salva no endereço do array (arr) passado na chamada da função. Assim esse array já recebe todos os valores requeridos.