



Funções

- O que são
 - Blocos de códigos independentes
 - Permitem organizar e centralizar rotinas

The diagram shows two vertical columns of code blocks, each containing several 'xxx' placeholders. Arrows point from specific blocks in these columns to a single block on the right, illustrating how functions are used to centralize code.

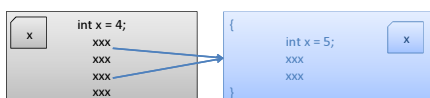
Funções

- Sintaxe
 - Cabeçalho
`tipo_de_retorno nome(parâmetros);`
 - Definição
`tipo_de_retorno nome(parâmetros)`
`{`
`// Bloco de código`
`}`

Características



- Escopo local
 - Variáveis locais são excluídas no final do bloco
 - Sem acesso a variáveis locais de outras funções
 - Acesso livre a variáveis globais
 - Variáveis do tipo **static** são compartilhadas entre diferentes chamadas da mesma função



Parâmetros e retorno



- Parâmetros
 - Variáveis locais
 - Excluídas no final do bloco
- Retorno
 - Tipos de retornos
 - Tipos de dados disponíveis no C
 - Ponteiro
 - Nada (**void**)
 - Instrução **return**

Chamadas



- Por valor
 - Uma cópia do valor é criada na memória
 - Alterações são perdidas no final do bloco
- Por referência
 - Utilização de ponteiros
 - Uma cópia do endereço de memória é criado
 - Alterações são realizadas no mesmo endereço de memória informado, sendo mantidas
 - Exemplo: **scanf**

```
// Por valor
void soma(int p1)
{
    p1 = p1 + p1;
}

// Por referência
void somaPonteiro(int *p1)
{
    *p1 = *p1 + *p1;
}

int x = 4;
soma(x);           // 4
somaPonteiro(&x);  // 8
printf("%d", x);
```

O método main



- Parâmetros
 - **argc**: número de parâmetros, tipo **int**
 - **argv**: parâmetros, array de string
 - Parâmetro ZERO é sempre o nome do programa
 - Padrão ANSI
- Retorno
 - Pelo padrão ANSI deve retornar **int**

```
int main (int argc, char *argv[])
{
    // Código
    return 0;
}
```

Interagindo com arrays / matrizes



- Passando como parâmetro

```
void imprima(int *matriz)
{
    int x;

    for(x=0; x<4; x++)
    {
        printf("%d ", matriz[x]);
    }
}
```

```
void imprima(int matriz[4])
{
    int x;

    for(x=0; x<4; x++)
    {
        printf("%d ", matriz[x]);
    }
}
```

```
void imprima(int matriz[])
{
    int x;

    for(x=0; x<4; x++)
    {
        printf("%d ", matriz[x]);
    }
}
```

```
int meusNumeros[4];

meusNumeros[0] = 15;
meusNumeros[1] = 47;
meusNumeros[2] = 62;
meusNumeros[3] = 84;

imprima(meusNumeros);    // 15 47 62 84
```

Retornando um array / matriz



- Envia-se um parâmetro adicional

```
void imprima(int *matriz)
{
    int x;

    for(x=0; x<4; x++)
    {
        printf("%d ", matriz[x]);
    }
}
```

```
void inverte(int *original, int *invertido)
{
    int x;

    for(x=0; x<4; x++)
    {
        invertido[4-1-x] = original[x];
    }
}
```

```
int meusNumeros[4], meusNumerosInvertidos[4];

meusNumeros[0] = 15;
meusNumeros[1] = 47;
meusNumeros[2] = 62;
meusNumeros[3] = 84;

inverte(meusNumeros, meusNumerosInvertidos);

imprima(meusNumeros);    // 15 47 62 84
imprima(meusNumerosInvertidos);    // 84 62 47 15
```

Parâmetro desconhecido



- Quando o parâmetro pode ser de diferentes tipos de dados dependendo da ocasião
- Parâmetro do tipo `void*`
- Pode ser utilizado como parâmetro da função e como seu retorno

```
void grave(void* buffer)
{
    FILE* arquivo;
    arquivo = fopen("dados.txt", "wb");
    fwrite(&buffer, sizeof(buffer), 1, arquivo);
    fclose(arquivo);
}
```

Parâmetros variáveis



- Funções com número e tipos de parâmetros indefinidos
- Exemplo: `printf("%d %d %d", 35, 42, 66);`
- Instrução reticências (...)
- Pelo menos um parâmetro deve ser fixo
- Biblioteca **stdarg.h**
 - **va_list**: matriz de argumentos
 - **va_start(matriz, primeiro parâmetro)**: inicializa a matriz
 - **va_arg(matriz, tipo de dado)**: acessa a matriz
 - **va_end(matriz)**: encerra o uso

Ponteiro para função




- Cada função possui seu ponto de entrada
 - Endereço de memória no código-objeto
- Sabendo o ponto de entrada, pode-se apontar para ele

```
int (*pf) (const char *);
puts("texto");
pf = puts;
pf("texto");
00003443("texto");
```

```
int puts(char* string)
{
    // Corpo da função
}
```

00003443

Recursão 

- Função que invoca ela mesma
- Looping infinito
- Exemplo: cálculo de fatorial de um número

```
int fatorial(int n)
{
    if(n != 1)
    {
        return n * fatorial(n-1);
    }
    else
    {
        return 1;
    }
}

int x = fatorial(4);
printf("%d", x); // 24
```

