

Linguagem C Função

Prof. Edwar Saliba Júnior
Fevereiro de 2011

Conceitos

- As técnicas de programação dizem que, sempre que possível, evite códigos extensos, separando o mesmo em funções, visando um fácil entendimento e uma manutenção facilitada;
- De acordo com a técnica, devem-se agrupar códigos correlatos em uma função;
- Uma outra utilização de função é quando um trecho de código será utilizado muitas vezes no programa. Deve-se colocar este trecho em uma função e sempre que for preciso chamar a função;
- A Linguagem C possibilita criar funções, sendo possível passar parâmetros para elas e retornar valores tanto no nome da função como em algum parâmetro passado;
- Enfim, lembre-se que: tudo em C é função.

Função e Protótipo (Assinatura da Função)

- Para ser usada, uma função deve estar previamente definida, isto é, deve-se indicar para o compilador qual o nome da função e quais são os parâmetros esperados;
- Uma maneira simples de se resolver isto, é a colocação da função antes de seu uso;
- Quando se têm sistemas grandes, não é recomendável ter um único arquivo fonte, pois a manutenção seria impraticável;
- Neste caso, é possível ter uma função definida em um programa fonte e seu uso em outro programa fonte. Para resolver este problema a Linguagem C criou uma definição chamada de **protótipo** (assinatura da função);
- No protótipo de uma função é definido somente o necessário para o compilador não acusar erros. A definição do protótipo geralmente é colocada dentro de arquivos *header* (.h) e incluída dentro dos programas fontes;
- No protótipo somente são informados o nome da função, o seu tipo de retorno e o tipo de cada parâmetro esperado.



Tipos de Funções

- **Funções pré-definidas pela linguagem:**
 - Funções da biblioteca;
 - Definidas nos arquivos da biblioteca da linguagem;
- **Funções definidas pelo programador:**
 - Escritas pelos programadores nos programas.

Por quê criar funções?

- **Para reduzir a complexidade de um programa:**
 - Elaborar a solução em partes pequenas e bem definidas;
 - Uma tarefa complexa é dividida em funções mais simples;
- **Para evitar a repetição de código ao longo do programa:**
 - Diminuir o tamanho do código;
 - Menos erros, menor custo de manutenção, menor tempo de programação;
- **Reutilização de código:**
 - Código já testado, sem erros => custo menor de programação e maior confiabilidade.

Características das Funções

- **Uma função deve realizar uma única tarefa bem definida;**
- **Toda função tem um nome único. De preferência bem significativo:**
 - **Serve para que a função seja invocada (chamada);**
- **Uma função pode ser invocada a partir de outras funções:**
 - **Por exemplo, a partir de *main()*;**
- **Uma função deve ser programada de forma a fazer exatamente o que se espera dela, sem efeitos colaterais.**

Características das Funções

- O código da função deve ser independente do programa e deve ser tão genérico quanto possível:
 - Para que possa ser utilizado em outros programas;
- Uma função pode receber parâmetros de execução, para se adaptar a situações distintas e ser genérica;
- Uma função pode retornar um valor como resultado de seu trabalho:
 - comando *return*;
- Uma função é constituída por instruções em C. De acordo com a sintaxe da linguagem:
 - Em C não se pode definir funções dentro de funções, portanto, todas as funções estão no mesmo nível.

Funções e Escopo

- Sintaxe:

```
tipoValorDeRetorno nomeDaFunção (parâmetros) {  
    declarações;  
    comandos;  
}
```

- Escopo:

- região de um programa em que um identificador é válido;
- o corpo da função compõe um escopo;
- identificadores definidos no corpo da função são **locais**;
- parâmetros são identificadores **locais**:
 - inicializados pelos argumentos passados na chamada da função.

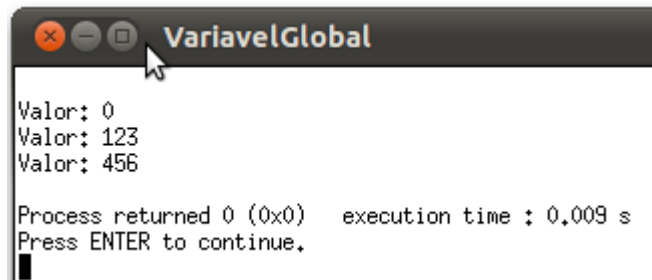
Escopo de Variáveis

- Entende-se como escopo de variáveis, a área onde o valor e o nome dela tem significado no código-fonte.
- Pode-se ter dois tipos de variáveis na Linguagem C:
 - Global - quando a variável é definida fora de qualquer função. Esta variável pode ser usada em qualquer função e o significado dela abrange todo o programa fonte, a partir de sua declaração.
 - Locais - são definidas dentro de funções e o seu significado é válido somente dentro da função que foi declarada. Assim, pode-se ter duas variáveis com o mesmo nome em funções diferentes.

Exemplo de Escopo de Variáveis

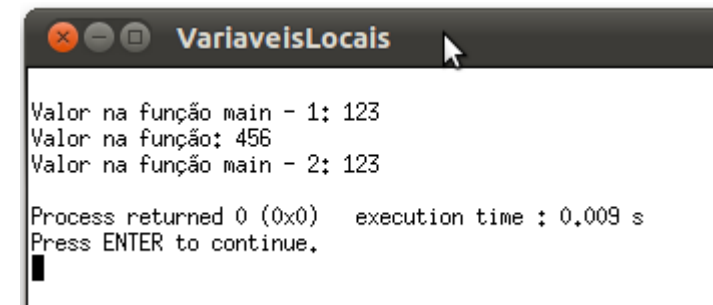
- Global e Local

```
1 #include <stdio.h>
2
3 int valor = 0;
4
5 void alteraValor(){
6     valor = 456;
7 }
8
9 int main()
10 {
11     printf("\nValor: %i", valor);
12     valor = 123;
13     printf("\nValor: %i", valor);
14     alteraValor();
15     printf("\nValor: %i", valor);
16     printf("\n");
17
18     return 0;
19 }
20
```



```
VariavelGlobal
Valor: 0
Valor: 123
Valor: 456
Process returned 0 (0x0)   execution time : 0,009 s
Press ENTER to continue.
```

```
1 #include <stdio.h>
2
3 void alteraValor(){
4     int valor = 456;
5     printf("\nValor na função: %i", valor);
6 }
7
8 int main()
9 {
10     int valor = 123;
11     printf("\nValor na função main - 1: %i", valor);
12     alteraValor();
13     printf("\nValor na função main - 2: %i", valor);
14     printf("\n");
15
16     return 0;
17 }
18
```



```
VariaveisLocais
Valor na função main - 1: 123
Valor na função: 456
Valor na função main - 2: 123
Process returned 0 (0x0)   execution time : 0,009 s
Press ENTER to continue.
```

Funções Retornando Valores

- Exemplo:

```
int power(int base, int exp){  
    int i, p = 1;  
    for(i = 1; i <= exp; ++i){  
        p *= base;  
    }  
    return(p);  
}
```

- O valor retornado por uma função é sempre copiado para o contexto de chamada.

```
X = power(2,5);           // Atribuição.  
if(power(7,6) > 12345)   // Comparação.  
    printf("Número muito grande!");  
x = 10 * power(3,9);     // Expressão.  
vetor[power(2,2)];       // Índice.  
funcao(power(4,9), valor); // Argumento.
```

Exemplo – Cálculo de Potência

- Faça um programa que imprima na tela, o resultado dos seguintes cálculos 2^9 , 3^5 e 7^7 .

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i, total1 = 1, total2 = 1, total3 = 1;
7
8      for(i = 0; i < 9; i++)
9          total1 = total1 * 2;
10
11     for(i = 0; i < 5; i++)
12         total2 = total2 * 3;
13
14     for(i = 0; i < 7; i++)
15         total3 = total3 * 7;
16
17     printf("\n0 resultado da potência de 2^9 = %i", total1);
18     printf("\n0 resultado da potência de 3^5 = %i", total2);
19     printf("\n0 resultado da potência de 7^7 = %i", total3);
20
21     return 0;
22 }
```

O mesmo código repetido três vezes. Péssima prática de programação!

Função para Cálculo de Potência

- Vamos resolver novamente, o exemplo anterior. Desta vez usaremos função.

```
1  #include <stdio.h>
2
3  int potencia(int base, int exp){
4      int i, total = 1;
5
6      for(i = 0; i < exp; i++)
7          total = total * base;
8
9      return(total);
10 }
11
12 int main()
13 {
14     int total1 = 1, total2 = 1, total3 = 1;
15
16     total1 = potencia(2,9);
17     total2 = potencia(3,5);
18     total3 = potencia(7,7);
19
20     printf("\n0 resultado da potência de 2^9 = %i", total1);
21     printf("\n0 resultado da potência de 3^5 = %i", total2);
22     printf("\n0 resultado da potência de 7^7 = %i", total3);
23
24     return 0;
25 }
26
```



Passagem de Parâmetros em C

- Por **valor** ou **cópia**:
 - os parâmetros contêm uma cópia válida do argumento correspondente:
 - esta cópia é válida no escopo da função.
 - o argumento pode ser uma variável ou o endereço de uma variável;
 - mudanças feitas nos parâmetros são feitas na cópia local e não alteram os argumentos;
 - a função pode acessar uma variável externa quando o seu endereço (da variável) for passado com argumento;
 - quando a função termina os valores deixam de existir:
 - fim do escopo.

Passagem de Parâmetros

- Em C os argumentos de uma função são sempre passados por valor:
 - isto significa que uma cópia do valor do argumento é passado para a função.

```
void contador(int i) {  
    printf("i: ");  
    while(i < 10)  
        printf("%d ", i++);           // i: 2 3 4 5 6 7 8 9  
}  
  
void main() {  
    int i = 2;  
    contador(i);  
    printf("i = %d ", i);           // i = 2  
}
```

Passagem de Parâmetros

- Como mudar o valor de uma variável externa, dentro da função?
 - passando o endereço do argumento para a função, pois, tendo o endereço, acessamos diretamente a posição de memória.
- Observações importantes:
 - alguns autores de livros de programação em C chamam este tipo de passagem de valores de: “passagem por referência”;
 - no entanto, como já foi dito, em só existe passagem de valores por **valor** ou **cópia**;
 - mesmo quando o endereço da variável (argumento) é passado.
 - o mecanismo de “passagem por referência” existe em algumas linguagens de programação: Pascal, C++, dentre outras.

Passagem de Parâmetros

- Passando o endereço do argumento:

```
void contador(int *i) {
    printf("i: ");
    while(*i < 10)
        printf("%d ", (*i)++);    // i: 2 3 4 5 6 7 8 9
}

void main() {
    int i = 2;
    contador(&i);
    printf("i = %d ", i);        // i = 10
}
```

Exemplo – Função e Protótipo

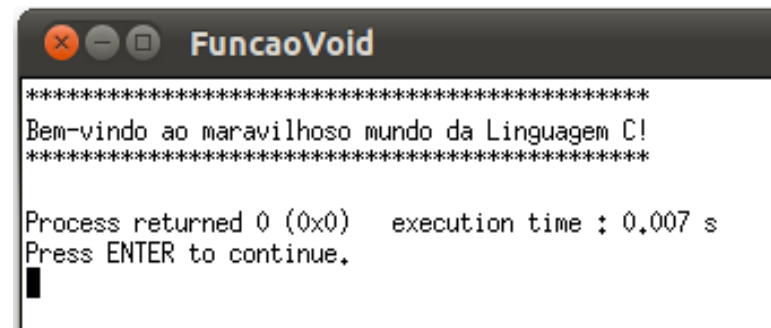
- Poderíamos ter definido a função “potencia”, depois da função “main”.
- Mas para isto, teríamos que ter definido o protótipo da função “potencia”, para não termos um erro de compilação.

```
1  #include <stdio.h>
2
3  int potencia(int base, int exp);
4
5  int main()
6  {
7      int total1 = 1, total2 = 1, total3 = 1;
8
9      total1 = potencia(2,9);
10     total2 = potencia(3,5);
11     total3 = potencia(7,7);
12
13     printf("\n0 resultado da potência de 2^9 = %i", total1);
14     printf("\n0 resultado da potência de 3^5 = %i", total2);
15     printf("\n0 resultado da potência de 7^7 = %i", total3);
16
17     return 0;
18 }
19
20 int potencia(int base, int exp){
21     int i, total = 1;
22
23     for(i = 0; i < exp; i++)
24         total = total * base;
25
26     return(total);
27 }
```

Função com Tipo de Retorno *void*

- Uma função pode não ter que retornar valor algum;
- Para este tipo de função, define-se o tipo de retorno como *void*, e neste caso não se usa o comando “*return*” no final da função;
- Em algumas linguagens de programação, este tipo de função é conhecida como “procedimento”.

```
1  #include <stdio.h>
2
3  void imprimeLinha(){
4      int i;
5
6      for (i=1; i<=46; i++)
7          putchar('*');
8      putchar('\n');
9  }
10
11 int main()
12 {
13     imprimeLinha();
14     printf("Bem-vindo ao maravilhoso mundo da Linguagem C!\n");
15     imprimeLinha();
16     return 0;
17 }
18
```



```
FuncaoVoid
*****
Bem-vindo ao maravilhoso mundo da Linguagem C!
*****

Process returned 0 (0x0)   execution time : 0,007 s
Press ENTER to continue.
```

Passagem de Parâmetros para Funções

- Por valor e Por referência (**Simulação**)

```
1 #include <stdio.h>
2
3 float soma(float a, float b){
4     return(a + b);
5 }
6
7 int main()
8 {
9     float val1 = 0, val2 = 0, total = 0;
10
11     printf("Digite o primeiro valor: ");
12     scanf("%f",&val1);
13     printf("Digite o segundo valor: ");
14     scanf("%f",&val2);
15
16     total = soma(val1,val2);
17
18     printf("O total da soma é: %f",total);
19
20     return 0;
21 }
22
```

```
1 #include <stdio.h>
2
3 void soma(float a, float b, float *tot){
4     *tot = a + b;
5 }
6
7 int main()
8 {
9     float val1 = 0, val2 = 0, total = 0;
10
11     printf("Digite o primeiro valor: ");
12     scanf("%f",&val1);
13     printf("Digite o segundo valor: ");
14     scanf("%f",&val2);
15
16     soma(val1,val2, &total);
17
18     printf("O total da soma é: %f",total);
19
20     return 0;
21 }
22
```

Exemplo Clássico

- A função “troca”:
 - implementar uma função `troca(x, y)` que recebe como parâmetros dois inteiros e troca seus valores:
 - `x` deve ficar com o valor de `y` e vice-versa.

- exemplo:

```
int x = 10, y = 20;  
troca(x, y);  
printf("x = %d e y = %d", x, y);
```

- deve produzir:

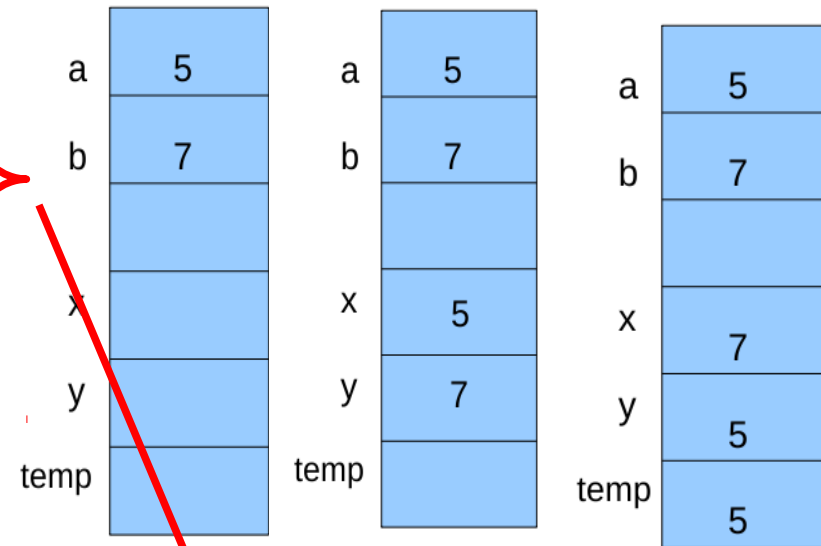
```
x = 20 e y = 10
```

Função Troca

- Implementação:

```
void troca(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}  
  
void main() {  
    int a = 5, b = 7;  
    troca(a, b);  
    printf("a = %d e b = %d", a, b);  
}
```

Memória do computador
em três instantes.



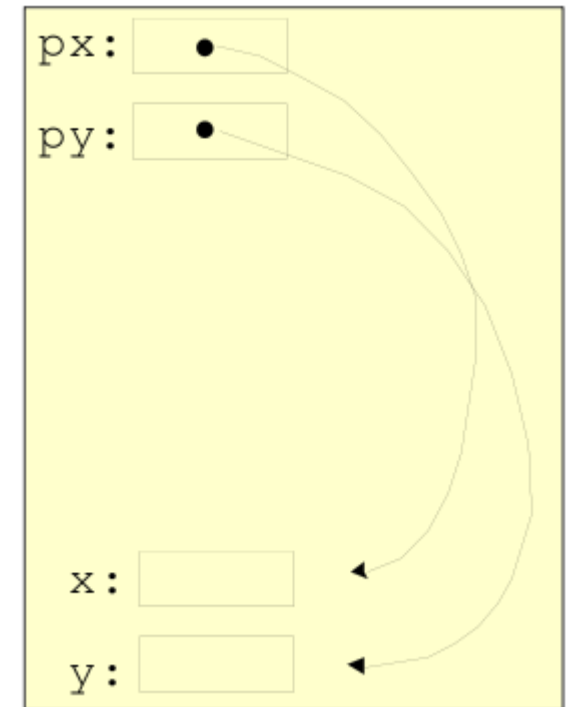
Não funciona!
"troca" copia os valores,
não altera as variáveis
originais (a e b).

Função Troca

- Implementação utilizando endereço de memória dos argumentos:

```
void troca(int *px, int *py){
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}

void main(){
    int a = 5, b = 7;
    troca(&a, &b);
    printf("a = %d e b = %d", a, b);
}
```



Passando o endereço dos argumentos e estando a função "troca" preparada para isto, a troca é efetivada.



Passagem de Parâmetros

- Conclusão: quando é passar valor ou o endereço do argumento?
 - se a variável for um vetor o endereço é sempre passado, pois, o nome do vetor corresponde ao seu endereço;
 - se a variável não precisa ou não deve ser alterada na função chamada, então:
 - passar o valor da variável.
 - se a variável precisa ser alterada pela função chamada e a mesma está preparada para isto:
 - passar o endereço da variável.

Função Copiar *String*

Versão em vetor para iniciantes

```
//copia um string t para string s
void copia_str(char *s, char *t){
    int i;
    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}
```

Versão para programadores experientes

```
//copia um string t para string s
void copia_str(char *s, char *t){
    while ((*s = *t) != '\0'){
        s++;
        t++;
    }
}
```

Versão para "feras"

```
//copia um string t para string s
void copia_str(char *s, char *t){
    while (*s++ = *t++);
}
```

Passando Parâmetros para `main()`

- É possível passar parâmetros diretamente para a função `main` a partir do prompt de comando;

- Exemplos:

```
/home/aluno> factorial 5
```

```
/home/aluno> copia arq1 arq2
```

- O que é preciso fazer?
 - definir os parâmetros em `main`;
 - ler os parâmetros na função `main`;
 - utilizá-los no programa.
- `main()` é definida com dois parâmetros:

```
main(int argc, char *argv[])
```

`argc` = número de argumentos passados;

`argv` = vetor de argumentos (*string*).

Parâmetros para `main()`

- Convenções:
 - os nomes são lidos como *strings*;
 - o nome do programa é o primeiro argumento:
 - assim, `argc` é no mínimo 1;
 - e este se encontra na posição `argv[0]`.
 - os demais argumentos, se existirem, se encontram nas posições: `argv[1]` até `argv[argc-1]`;
 - o padrão define a posição `argv[argc]` que conterá o valor `NULL`.

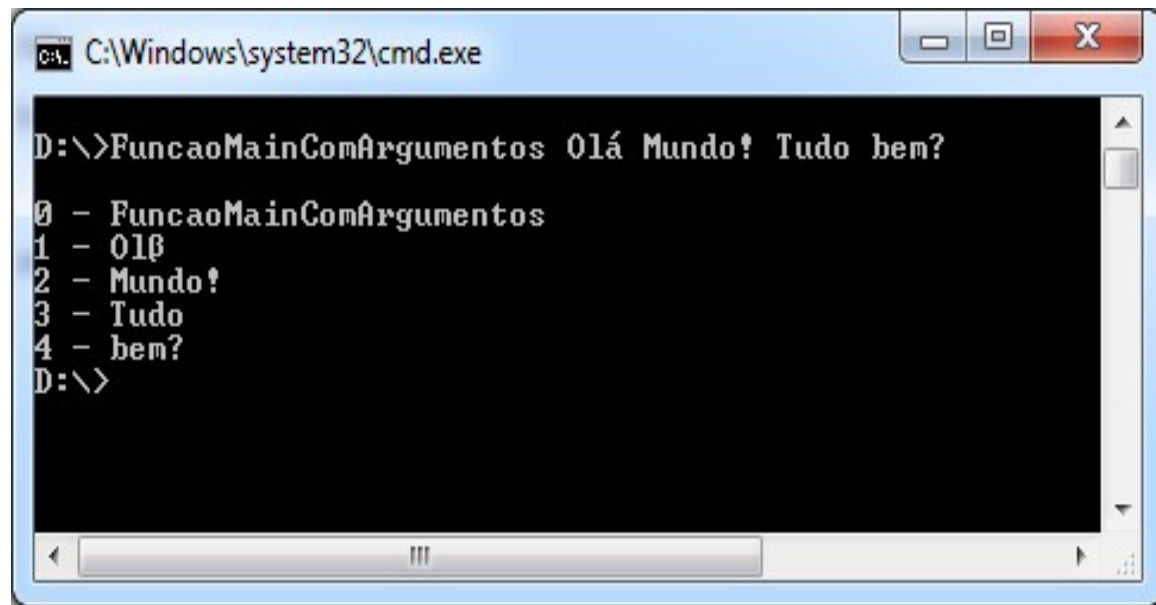
- Exemplo:

```
echo Hello, world!
```

```
argc = 3
```

Exemplo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  |
4  int main(int argc, char *argv[])
5  {
6      int i = 0;
7
8      for(i = 0; i < argc; i++){
9          printf("\n%d - %s", i, argv[i]);
10     }
11
12     return 0;
13 }
14
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "D:\>". The user has entered the command "FuncaoMainComArgumentos Olá Mundo! Tudo bem?". The output of the program is displayed as follows:

```
D:\>FuncaoMainComArgumentos Olá Mundo! Tudo bem?
0 - FuncaoMainComArgumentos
1 - Olá
2 - Mundo!
3 - Tudo
4 - bem?
D:\>
```



Bibliografia

- LAUREANO, Marcos. **Programação em C para ambiente Linux**. Disponível em: <<http://br-c.org/doku.php>>. Acesso em: 06 fev. 2011.
- MURTA, Cristina Duarte. *Slides* da disciplina de Programação de Computadores I. CEFET-MG, 2010.
- SENNE, Edson Luiz França. **Primeiro Curso de Programação em C**. 2. ed. Florianópolis: Visual Books, 2006.
- SOARES, Gabriela Eleutério. *Slides* da disciplina de Programação de Computadores I. CEFET-MG, 2011.