Universidade Federal do Espírito Santo

Universidade Federal
do Espírito Santo

Programação II

Prof.ª Claudia Boeres (boeres@inf.ufes.br)

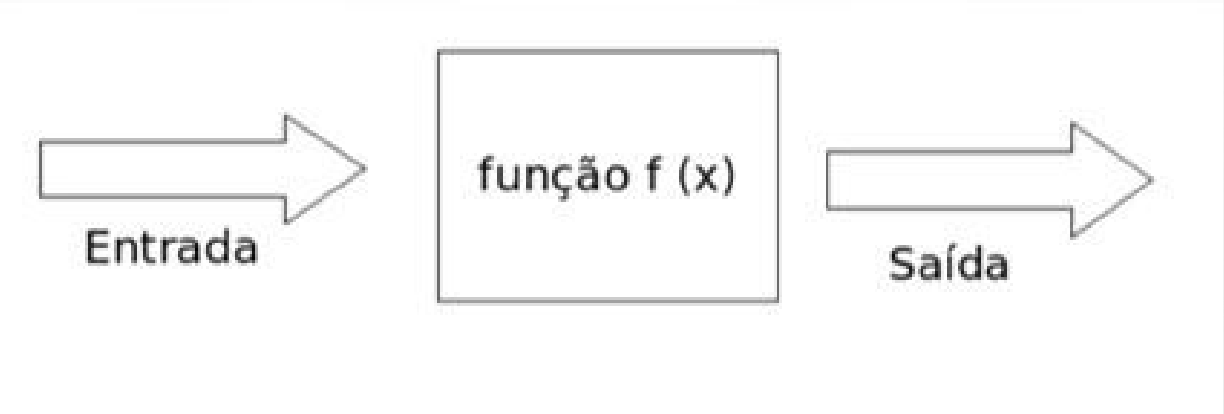
CT IX - Sala 206
Departamento de Informática
Centro Tecnológico
Universidade Federal do Espírito Santo

Modularização

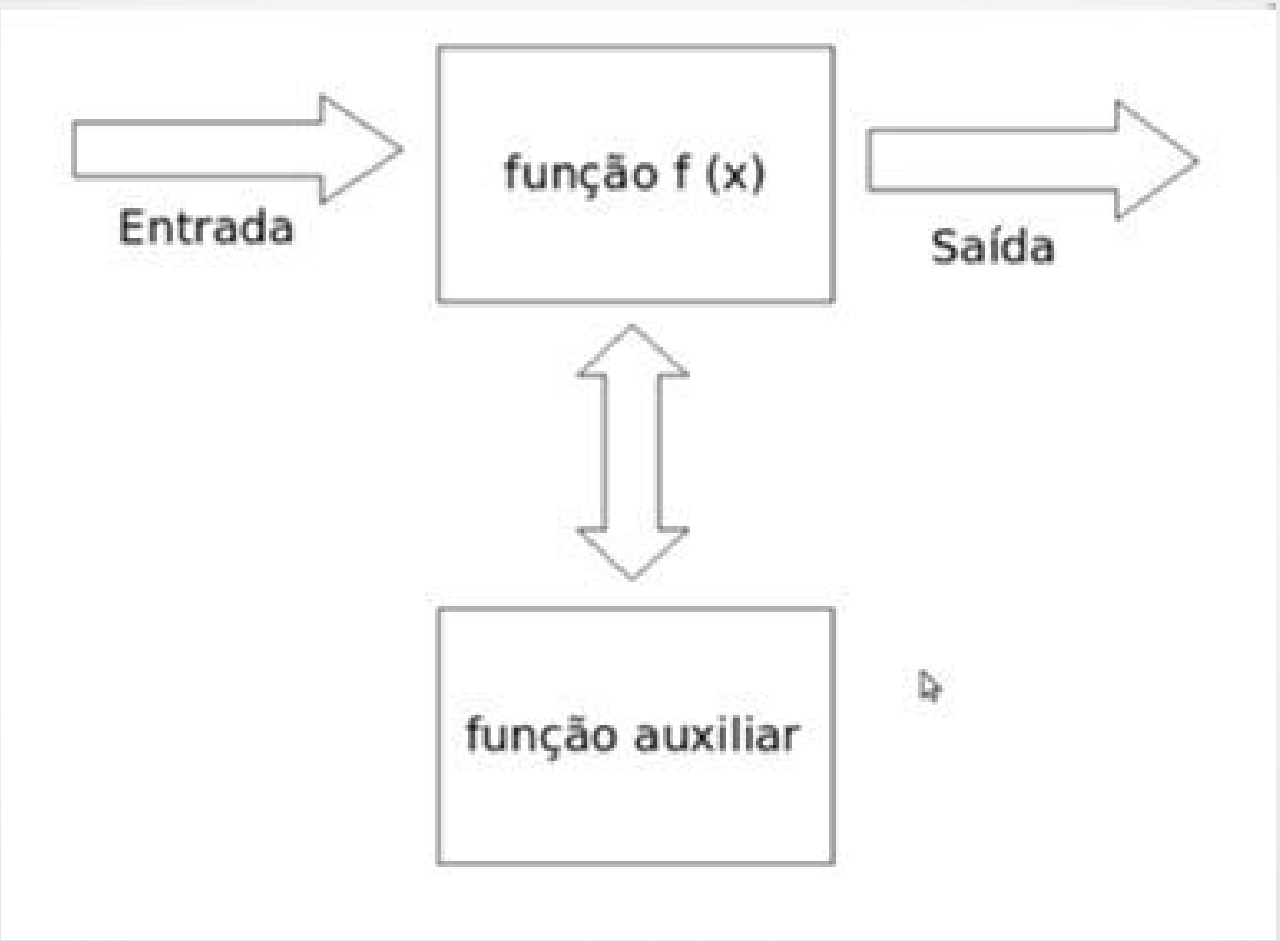
Definição

- Subprograma ou módulo:

Trecho de um programa que realiza qualquer operação computacional. Em C o termo subprograma é conhecido como função.



Definição



Partes de uma função

```
1  float calculaMedia (float a, float b);
```

Exemplo 3.1: Cabeçalho de um subprograma na linguagem C.

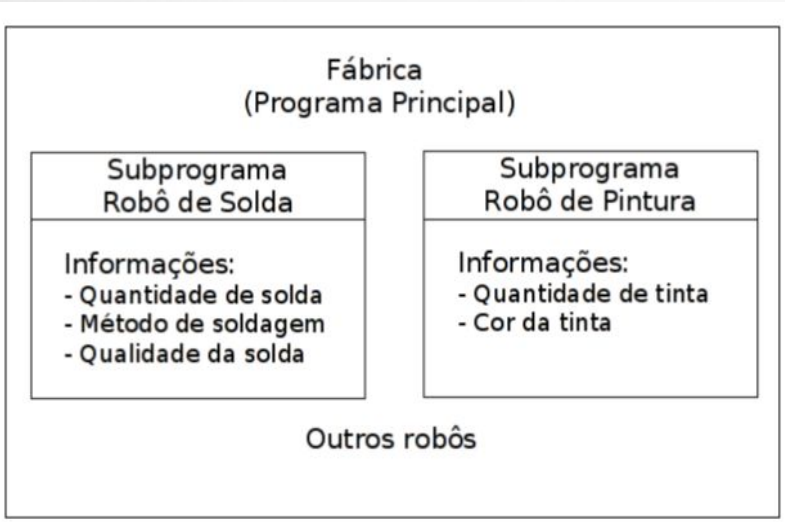


Figura 3.3: Dicionário de dados.

```
2⊕ * main.c
7
8⊖int roboSolda()
9 {
10      float refilSolda = 0; /*comprimento em m*/
11      int metodo = -1;
12      float qualidadeAtingida = 0; /*valor em %*/
13
14⊖             /*
15                Lógica
16              */
17
18      return 0;
19 }
20
21⊖int roboPintura(void)
22 {
23      float volumeTinta = 0; /*volume em ml*/
24      int cor = -1;
25
26⊖             /*
27                Lógica
28              */
29
30      return 0;
31 }
```

Partes de uma função

- **Corpo:** lógica do código, ou seja, a parte que efetivamente implementa as ações requeridas pela função;
- **Comentários:** explicação do código com objetivo de melhorar a legibilidade do mesmo. Pode ser feito em em linha, em bloco de linhas, fora da função, ou combinando todas essas formas.

Partes de uma função

```
 2⊕ * main.c
 7
 8⊖/*Função que implementa as operações que controlam o
 9  * robo de solda*/
10⊖int roboSolda()
11 {
12     /*Declaração das variáveis do robo de solda*/
13     float refilSolda = 0; /*comprimento em m*/
14     int metodo = -1;
15     float qualidadeAtingida = 0; /*valor em %*/
16
17⊖            /*
18              Lógica
19             */
20
21     return 0;
22 }
23
24⊖/*Função que implementa as operações que controlam o
25  * robo de pintura*/
26⊖int roboPintura(void)
27 {
28     /*Declaração das variáveis do robo de pintura*/
29     float volumeTinta = 0; /*volume em ml*/
30     int cor = -1;
31
32⊖            /*
33              Lógica
34             */
35
36     return 0;
37 }
```

Exercício

Como definir uma função para calcular a distância entre dois pontos no plano cartesiano?

Chamadas de funções

- Quando uma função solicita serviços de outra função dizemos que foi feita uma chamada de função (subprograma);
- Durante a execução de um programa podem ser feitas diversas chamadas a uma função, ou seja, quantas forem necessárias;
- No entanto, ao chamar uma função é criada uma área de memória para o armazenamento das variáveis locais e esse procedimento é feito em tempo de execução;

Chamadas de funções: exemplo

```c
#include <stdlib.h>
#include <stdio.h>
/*É necessário incluir o comando -lm na chamada do link editor (linker)*/
#include <math.h>

float distanciaEuclidiana(float x1, float y1, float x2, float y2)
{
    return sqrt( pow(x1-x2,2) + pow(y1-y2,2) );
}

int main()
{
    float xa,ya,xb,yb,dist;

    printf("Forneça os pontos da reta, em m, no formato x1 y1 x2 y2 (ex:1 2 4 5): ");
    scanf("%f%f%f%f", &xa, &ya, &xb, &yb);

    dist = distanciaEuclidiana(xa, ya, xb, yb);

    printf("A distância entre os pontos é: %f\n", dist);

    /*Segunda chamada da função distanciaEuclidiana*/

    printf("\nForneça a localização das cidades A e B, em km, no formato xA yA xB yB (ex:100 450 1000 1300): ");
    scanf("%f%f%f%f", &xa, &ya, &xb, &yb);

    dist = distanciaEuclidiana(xa, ya, xb, yb);

    printf("A distância entre as cidades é: %f", dist);

    return 0;
}
```

Passagem de Parâmetros

- Para cada chamada da função com seus respectivos parâmetros de entrada é feita uma instanciação da mesma.

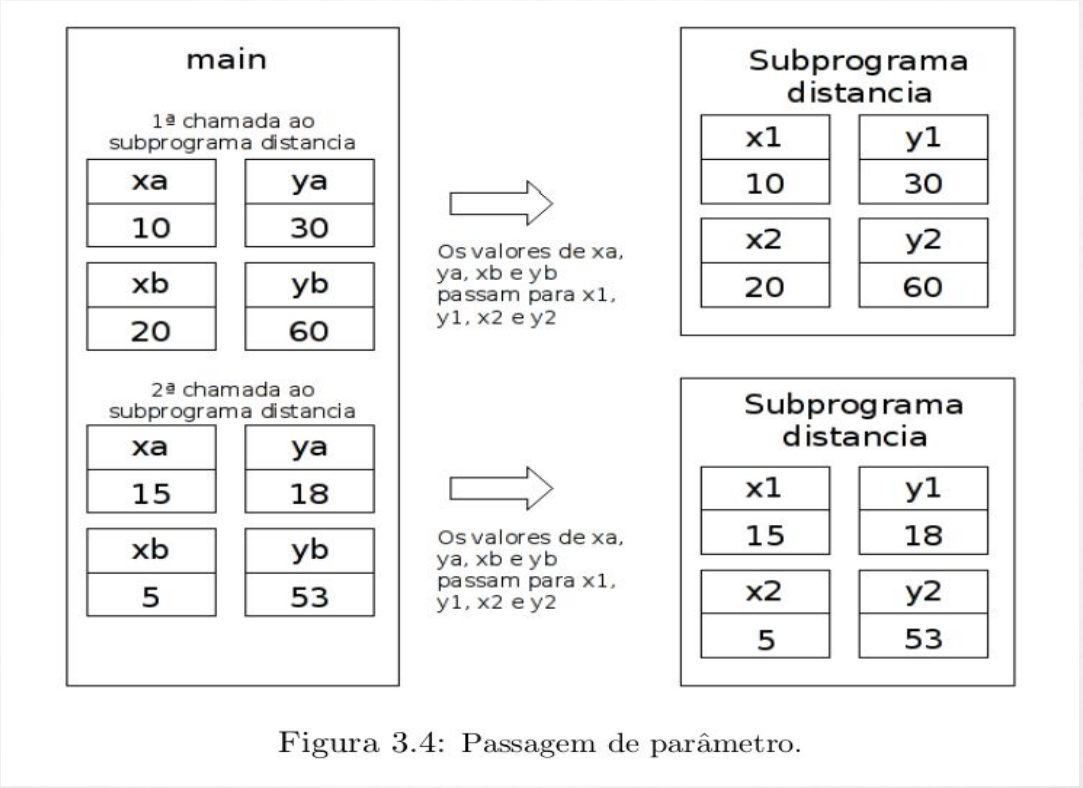


Figura 3.4: Passagem de parâmetro.

- Não é possível modificar o valor das variáveis da função que efetuou a chamada.

Retorno de Dados

- Cada função produz um valor final que deverá ser passado para a função que a chamou;
- O valor retornado pela função será atribuído a alguma variável da função que efetuou a chamada, ou então usado em alguma expressão;

Retorno de Dados

- Após a execução do comando **return** a execução da função termina, mesmo que existam mais instruções após o return.
- É possível ocorrer vários pontos de retorno em uma função.

```
1  int ehDivisor (int x, int y){
2      if (y == 0){
3          return 0;
4      }
5
6      if (x%y == 0){
7          return 1;
8      }else{
9          return -1;
10     }
11 }
```

Funções Especiais

- Existem funções que não retornam dados;
- Na linguagem C, para simbolizar essa situação usa-se o tipo **void** como tipo de retorno da função;

```
1  void multiplica (float a, float b, float c){
2
3          printf ("Resultado = %f", a*b*c);
4
5  }
```

Exemplo 3.15: Função sem retorno.

Funções Especiais

- Nem todas as funções precisam ter parâmetros de entrada;
- Assim, existem funções que executam suas instruções sem precisar de parâmetros de entrada;

```
1  int lerNumeros (void){
2     int x=0, temp, i;
3
4     for (i=0 ; i<5 ; i++){
5        printf ("digite um numero: ");
6        scanf ("%d", &temp);
7        x += temp;
8     }
9
10    return x;
11 }
```

Exemplo 3.14: Lista de parâmetros vazia

- O que acontece na compilação do código ao lado ?

```
 2⊕  * main.c
 7
 8⊕ /*Função que implementa as operações que controlam o
10⊖ int roboSolda()
11  {
12      /*Declaração das variáveis do robo de solda*/
13      float refilSolda = 0; /*comprimento em m*/
14      int metodo = -1;
15      float qualidadeAtingida = 0; /*valor em %*/
16
17⊖             /*
18               Lógica
19             */
20
21      return 0;
22  }
23
24⊕ /*Função que implementa as operações que controlam o
26⊖ int roboPintura(void)
27  {
28      /*Declaração das variáveis do robo de pintura*/
29      float volumeTinta = 0; /*volume em ml*/
30      int cor = -1;
31
32⊖             /*
33               Lógica
34             */
35
36      return 0;
37  }
38
39⊖ int main()
40  {
41
42      int aux;
43
44      aux = roboSolda(5.455454);
45      aux = roboPintura();
46      //aux = roboPintura(3.5646);
47
48      return 0;
49
50  }
```

Recursividade

- De forma geral a recursividade ocorre quando algo é definido a partir de si mesmo;
- Os elementos básicos para se definir uma recursão são:
  - Uma ou mais relações recursivas: quando o próximo elemento é definido a partir do anterior;
  - Uma ou mais bases de recursão: pontos de parada;

$$\text{lista} = \text{lista\_vazia} \Rightarrow \text{base da recursao}$$
$$\text{lista} = \text{elemento} + \text{lista} \Rightarrow \text{relação recursiva}$$

- Em programação, um método recursivo pode ser usado para definir uma função;

```
1  int fatorial (int n){
2      if (n == 0){
3          return 1;
4      }else{
5          return n * fatorial (n-1);
6      }
7  }
```

Recursividade

- As chamadas recursivas não geram código otimizado, porém um código recursivo pode ser a única forma de resolver um problema.
- Exemplo da implementação de chamadas recursivas:

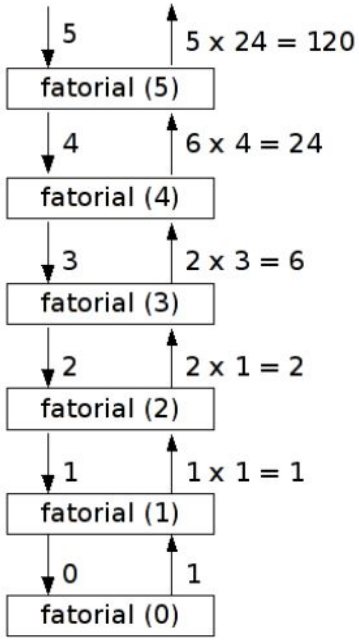


Figura 3.6: Fatorial utilizando recursão.

Exercícios

1. Faça uma função que calcule o fatorial de forma iterativa.

2. Faça uma função que calcula o mdc de forma iterativa e outra função que calcula o mdc de forma recursiva.

3. Faça uma função que retorne o maior de 3 números.

4. Faça uma função que imprima as raízes reais de uma equação de 2º grau.

Exercícios

5. Faça uma função que verifica se um número inteiro é palíndromo.

6. Faça uma função que verifica se, dados dois números inteiros positivos, se um deles é o segmento final do outro.

7. Dê um exemplo de função recursiva, implemente essa função em C, compile e teste-a;