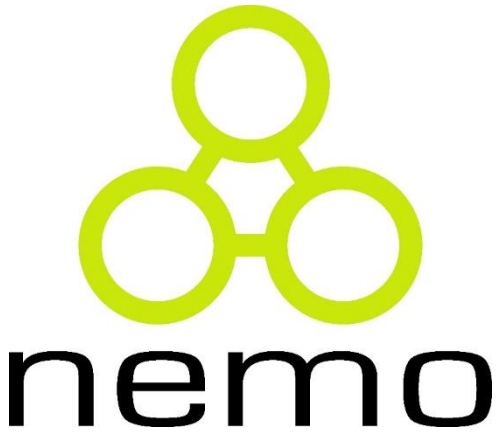


Programação 2

Jordana S. Salamon

jssalamon@inf.ufes.br

DEPARTAMENTO DE INFORMÁTICA

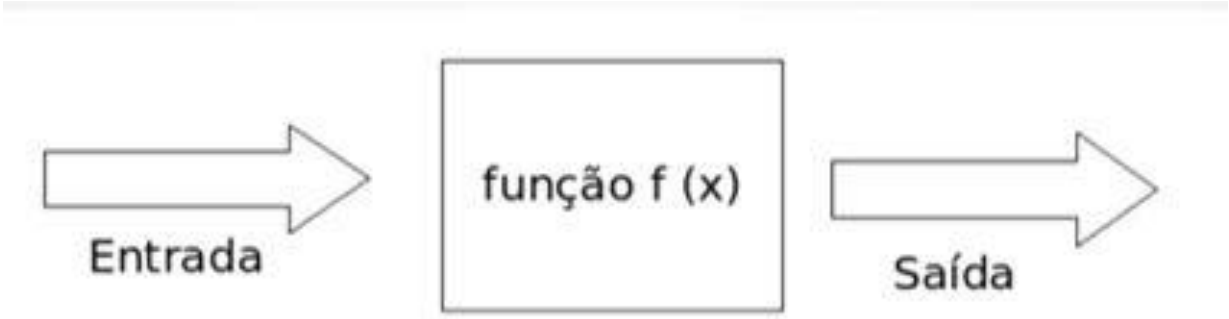
CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

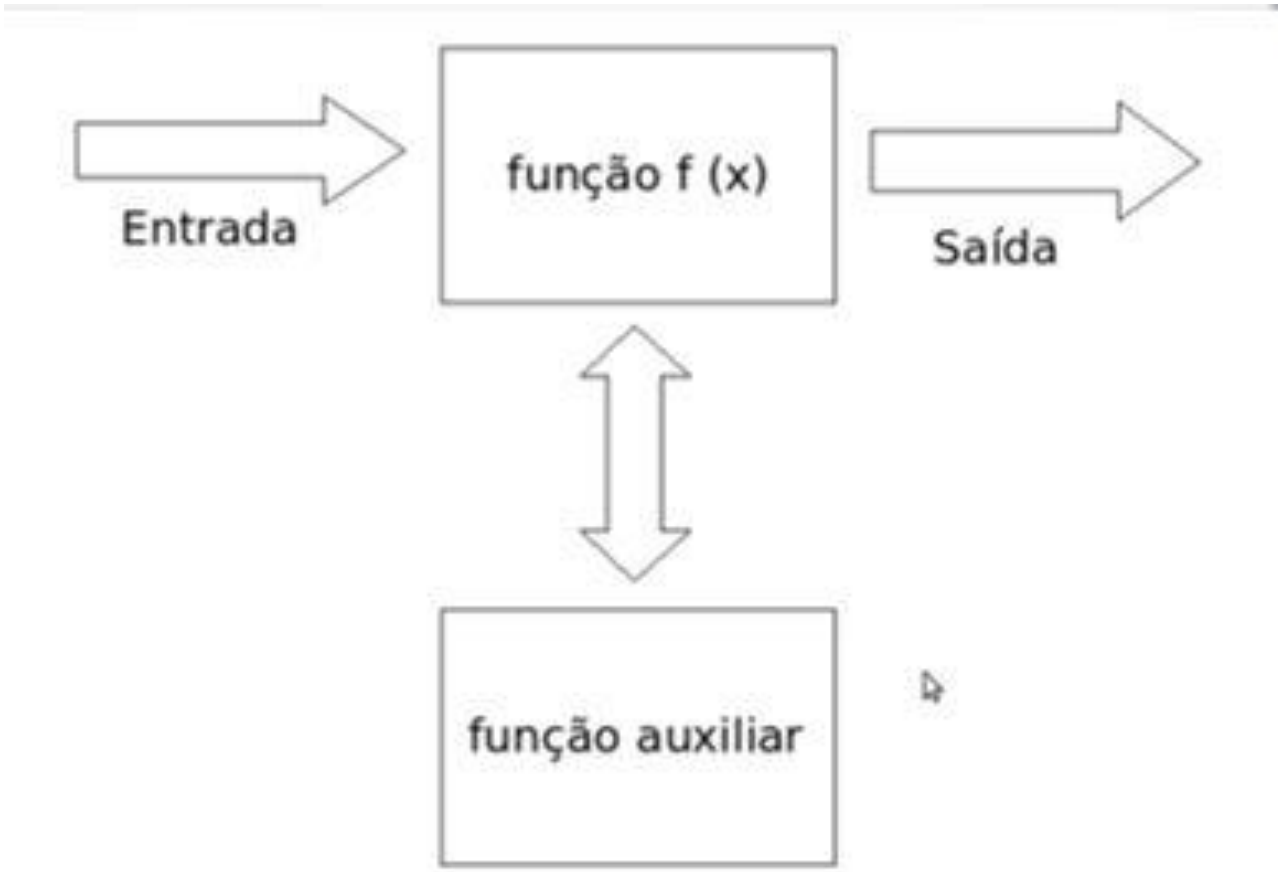
Modularização

Subprograma ou módulo:

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



Modularização



Modularização

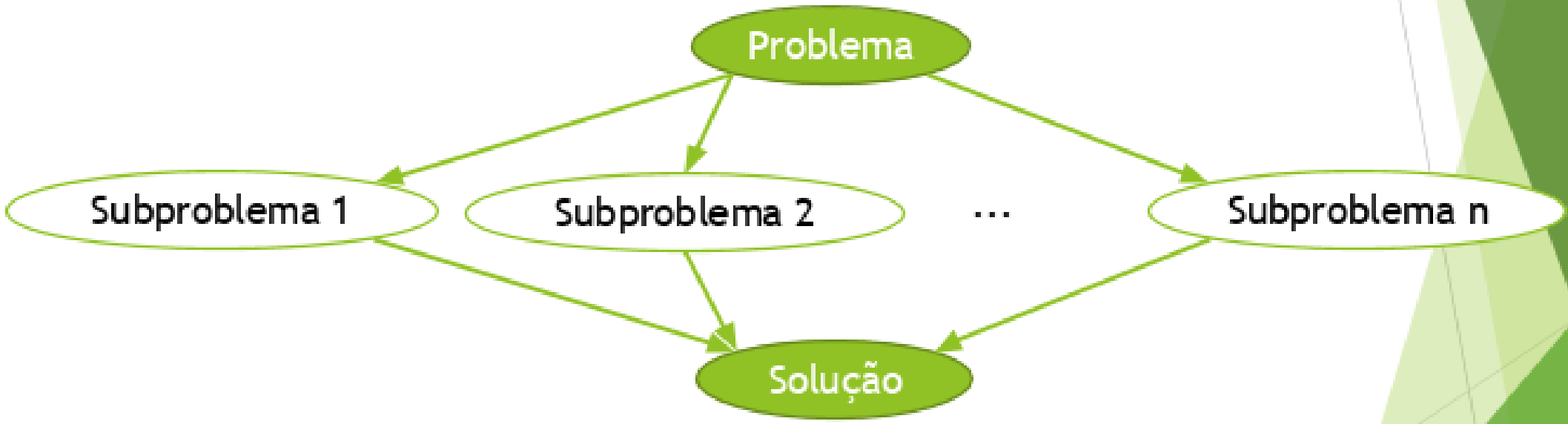
$$I_i = \int_{x_i}^{x_{i+1}} f(x)dx$$

?

nemo

Motivação da utilização de Funções

Dividir para Conquistar



nemo

Motivação de utilização de funções

- Subdivisão de algoritmos complexos
  - Facilidade de entendimento
- Estruturação de algoritmos
  - Facilidade de documentação e de detecção de erros
- Modularização de Sistemas
  - Facilidade de manutenção e reutilização

- Exemplos
  - Bibliotecas: fatorial, seno, cosseno, etc
  - Parte do programa que é repetida várias vezes
    - Verificar se o número é primo
    - Calcular média
    - Fatorial

Partes de uma função

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

Exemplo 3.1: Cabeçalho de um subprograma na linguagem C.

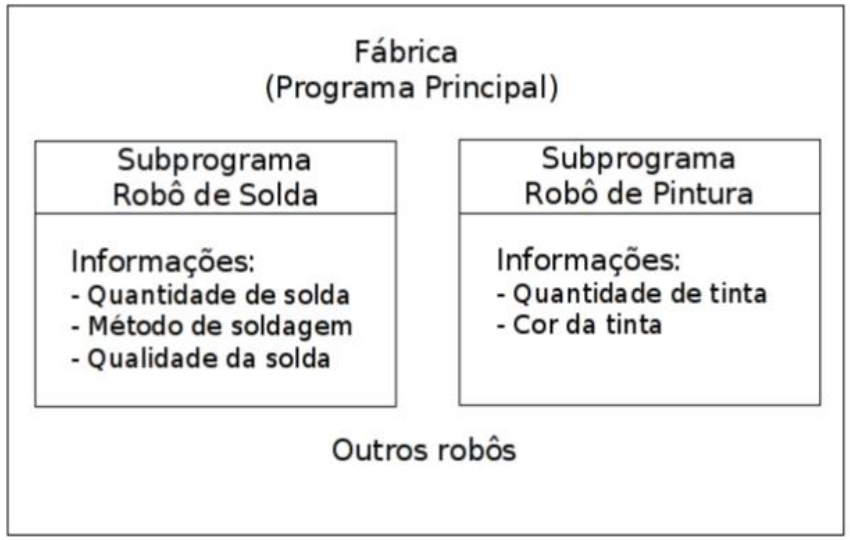


Figura 3.3: Dicionário de dados.

```c
 * main.c
 */

int roboSolda()
{
    float refilSolda = 0; /*comprimento em m*/
    int metodo = -1;
    float qualidadeAtingida = 0; /*valor em %*/

            /*
              Lógica
            */

    return 0;
}

int roboPintura(void)
{
    float volumeTinta = 0; /*volume em ml*/
    int cor = -1;

            /*
              Lógica
            */

    return 0;
}
```

nemo

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 linha, em bloco de linhas, fora da função, ou combinando todas essas formas.

Sintaxe

- Tipo do retorno
  - Define o tipo a ser retornado após o processamento
- Nome da função
  - Semelhante ao nome do algoritmo em pseudo-código
- Tipo e nome do parâmetro

```
Funcao <nome> (<parametros>) : <tipo_retorno>
Var <declaracao_variavel>
Inicio
      <codigos>
      retorna ...;
Fim
```

Pseudocódigo

```
<tipo_retorno> <nomeFuncao> (<parametros>) {
      <codigos>
      return ...;
}
```

C

nemo

Paralelo

```
Algoritmo calcula_fatorial
    var fat:inteiro
Inicio
      fat = fatorial(5)
      imprimir("Fatorial de 5 é: "+fat)
      fat = fatorial(10)
      imprimir("Fatorial de 10 é: "+fat)
Fim
Funcao fatorial(n:inteiro):inteiro
     var fat,i:inteiro
Inicio
     fat = 1
     Para i de 1 até n Passo 1 faça
          fat = fat * i
     fim_para
      retorne fat
Fim
```

```
#include<stdio.h>

int fatorial(int n){
      int fat,i;
      fat = 1;
      for(i = 1; i <=n; i++){
           fat = fat * i;
      }
      return fat;
}

int main(){
     int fat = fatorial(5);
     printf("Fatorial de 5 é: %d",fat);
     fat = fatorial(10);
     printf("Fatorial de 10 é: %d",fat);
     return 0;
}
```

Sintaxe

- Procedimentos
  - Não retornam um valor
  - void
- Funções
  - Retornam um valor como resultado do processamento
  - int, float, char, ...

Variáveis Globais e Locais

- Declaração de variáveis dentro da função

```
void função(int n){
    int x;
}
```

- Acesso a variáveis globais e locais
  - Variável definida em uma função não é acessível por outra

- Passagem de parâmetro por cópia
  - Alteração do parâmetro não interfere no valor original

Variáveis Locais

Só existem dentro da função fatorial

Só existe dentro da função main

```
#include<stdio.h>

int fatorial(int n){
      int fat,i;
      fat = 1;
      for(i = 1; i <=n; i++){
            fat = fat * i;
      }
      return fat;
}

int main(){
      int fat = fatorial(5);
      printf("Fatorial de 5 é: %d",fat);
      fat = fatorial(10);
      printf("Fatorial de 10 é: %d",fat);
      return 0;
}
```

nemo

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

```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;
}

```

nemo

Passagem de Parâmetros

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

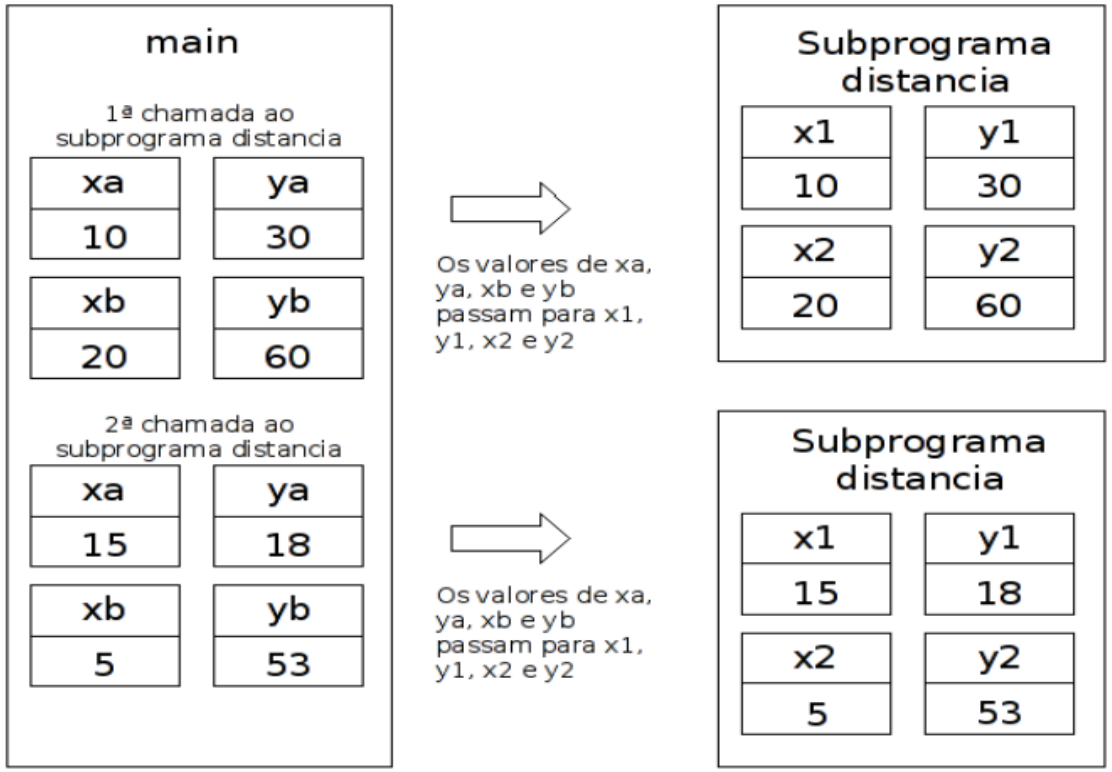


Figura 3.4: Passagem de parâmetro.

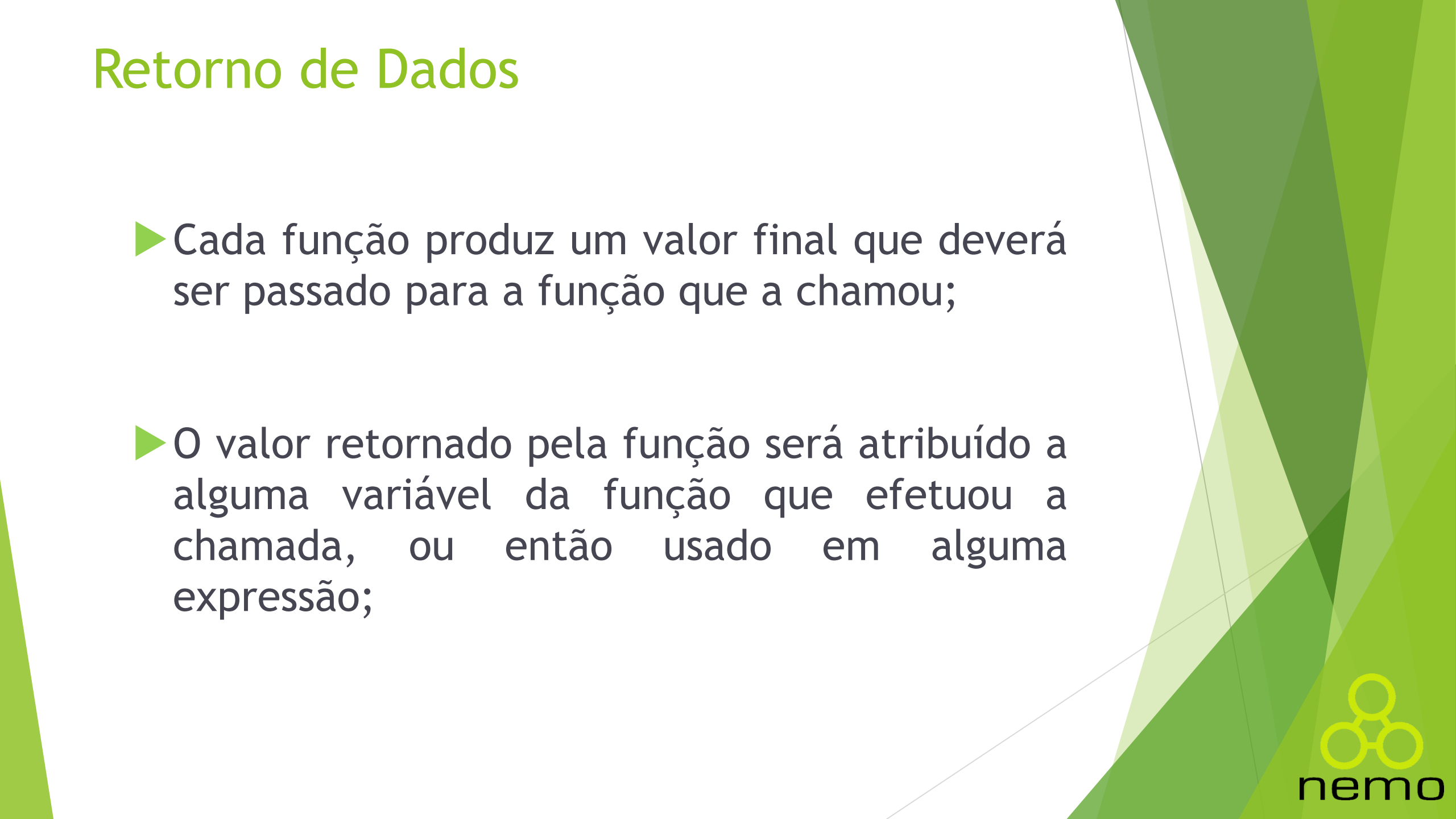
- Não é possível modificar o valor das variáveis da função que efetuou a chamada.

nemo

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 ocorrerem 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 }
```

nemo

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.

nemo

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

nemo

Exercícios

1. Escreva uma função que receba um número inteiro e retorne o seu antecessor.
2. Escreva uma função que receba 2 números inteiros e retorne o seu produto.
3. Escreva uma função que receba 3 números inteiros e retorne a sua média aritmética.
4. Escreva uma função que receba 3 números reais (ponto flutuante) e retorne a sua média aritmética.
5. Escreva uma função que receba 2 números inteiros, 2 números reais indicando pesos, e retorne a média ponderada.
6. Escreva uma função que receba dois valores inteiros representando, respectivamente, um valor de hora e um de minutos e retorne o valor equivalente em minutos.
7. Escreva uma função que receba quatro valores inteiros representando hora e minuto de dois horários, respectivamente, e retorne quantos minutos se passarem entre o primeiro e o segundo horário. Lembre-se que a primeira hora deve ser menor que a segunda.

nemo

That’s all Folks!