Linguagem C: Funções Parte 1*

Joice Otsuka

*Baseado no livro: Linguagem C Completa e Descomplicada, de André Backes

Função

- Funções são blocos de código que podem ser nomeados e chamados de dentro de um programa.

- Já usamos diversas funções
  - **printf()**: função que escreve na tela
  - **scanf()**: função que lê o teclado

Por que utilizar?

- Evitar que os blocos do programa fiquem grandes demais e mais difíceis de entender

- Facilitar a leitura do programa-fonte

- Separar o programa em partes(blocos) que possam ser logicamente compreendidos de forma isolada

Por que utilizar?

- Facilitar a estruturação e reutilização do código.
  - Permitem o reaproveitamento de código já construído
  - Evitam a repetição desnecessária de trechos de código que realizam a mesma tarefa, diminuindo assim o tamanho do programa e a ocorrência de erros

Função - estrutura

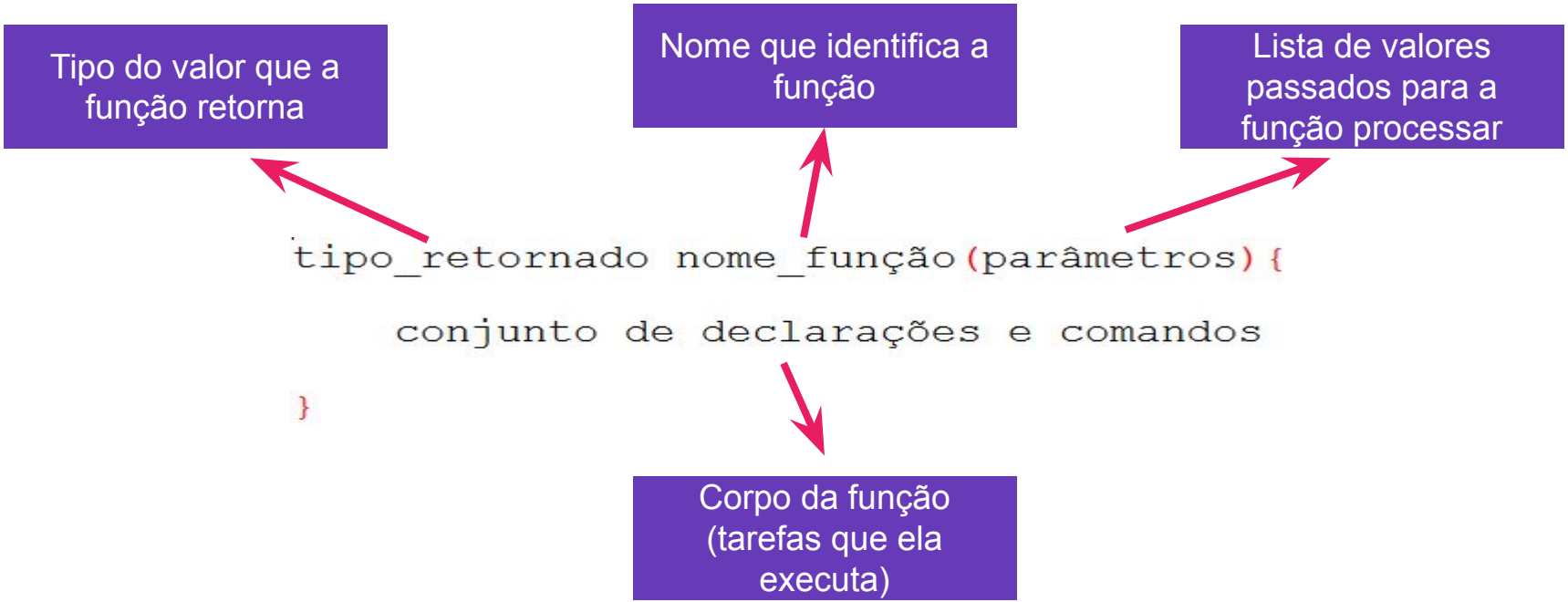
Uma função possui o seguinte formato:

```
tipo_retornado nome_da_funcao (tipo <parametro1>,
tipo <parametro2>,...,  tipo <parametro n>) {

    Comandos;

    return(valor de retorno);

}
```

Toda função tem um tipo, que determina o tipo do valor que será retornado. Caso não retorne valor, o tipo é **void**

Função - estrutura

- Forma geral de uma função:



Função - Corpo

- Formado pelos comandos que a função deve executar
- Processa os parâmetros (se houver), realiza outras tarefas e gera saídas (se necessário)

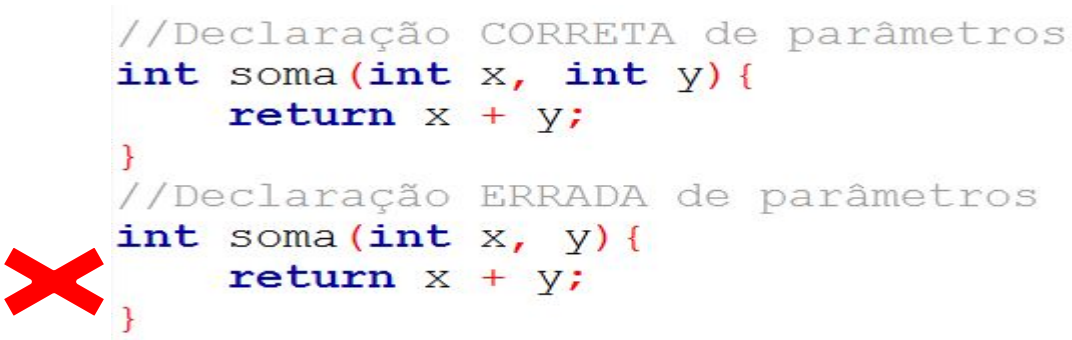
```
int soma(int x, int y){
    return x + y;
}
```

Função - Corpo

- Evita-se fazer operações de leitura e escrita dentro de uma função.
  - Uma função é construída com o intuito de realizar uma tarefa específica e bem definida
  - As operações de entrada e saída de dados (funções **scanf()** e **printf()**) geralmente são feitas em quem chamou a função (por exemplo, na **main()**)
  - Isso assegura que a função construída possa ser utilizada nas mais diversas aplicações, garantindo a sua generalidade

Função - Parâmetros

- A declaração de parâmetros é uma lista de variáveis juntamente com seus tipos:
  - *tipo1 nome1, tipo2 nome2, ... , tipoN nomeN*
  - Pode-se definir quantos parâmetros forem necessários, separados por vírgula



Função - Parâmetros

- É por meio dos parâmetros que uma função recebe informações do programa que a chamou
  - As variáveis recebidas como parâmetros não devem ser lidas dentro da função (caso contrário, o valor recebido será perdido)

```
int x = 2;
int y = 3;
```

```
int soma(int x, int y){
    return x + y;
}

int main(){
    int z = soma(2,3);

    return 0;
}
```

```
int soma(int x, int y){

    scanf("%d",&x);
    scanf("%d",&y);

    return x + y;
}
```

Função - Parâmetros

- Podemos criar uma função que não recebe nenhum parâmetro de entrada
- Isso pode ser feito de duas formas
  - Podemos deixar a lista de parâmetros vazia
  - Podemos colocar **void** entre os parênteses

```c
void imprime(){
    printf("Teste\n");
}


void imprime(void){
    printf("Teste\n");
}
```

Função - Retorno

- Uma função pode ou não retornar um valor
  - Se ela retornar um valor, alguém deverá receber este valor
  - Uma função que retorna nada é definida colocando-se o tipo **void** como valor retornado. **Funções que retornam nada são conhecidos como Procedimentos.**

Função - Retorno

- Podemos retornar qualquer valor válido em C
  - tipos pré-definidos: int, char, float e double
  - tipos definidos pelo usuário: struct
  - **Uma função não pode retornar um vetor**
    - A linguagem C não suporta a atribuição de um vetor a outro

Comando return

- O valor retornado pela função é dado pelo comando **return**

- Forma geral:
  - **return** *valor ou expressão*;
  - **return**;
    - Usada para terminar uma função que não retorna valor

- É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Comando return

Função com retorno de valor

```c
int soma(int x, int y){
    return x + y;
}

int main(){
    int z = soma(2,3);

    return 0;
}
```

Função sem retorno de valor (Procedimento)

```c
void imprime(){
    printf("Teste\n");
}

int main(){
    imprime();

    return 0;
}
```

Comando return

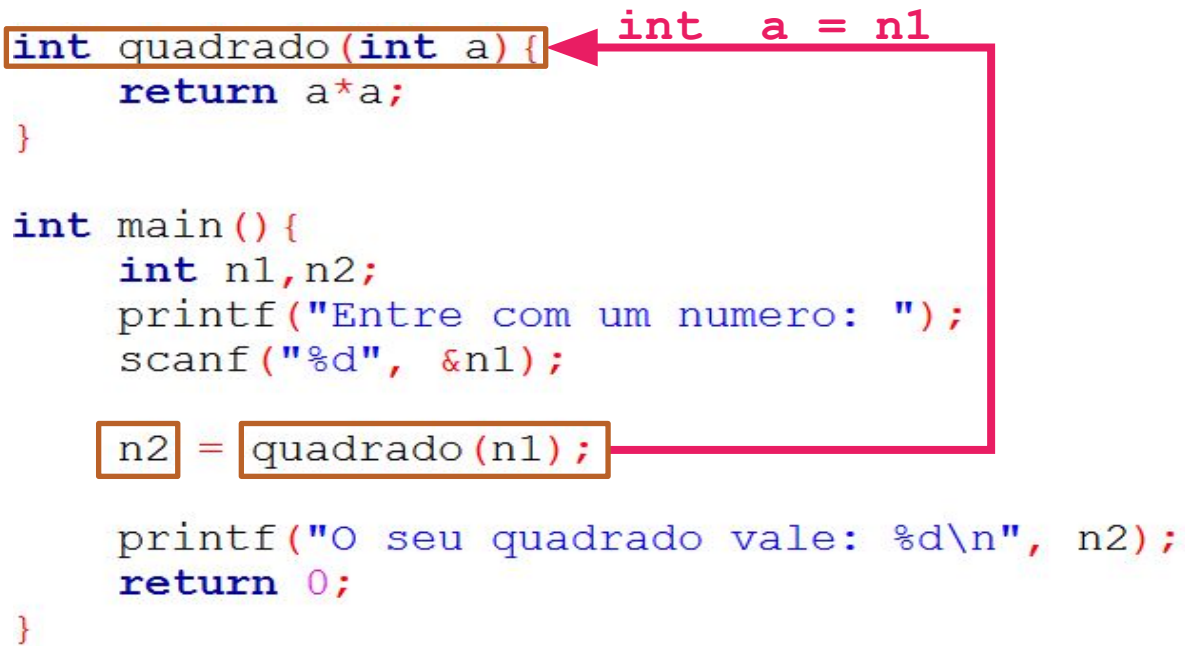
- Uma função pode ter mais de um comando **return**.
  - Quando o comando **return** é executado, a função termina imediatamente.
  - Todos os comandos restantes são **ignorados**.

```
int maior(int x, int y){
    if(x > y)
        return x;
    else
        return y;
    printf("Esse texto nao sera impresso\n");
}
```

ignorado

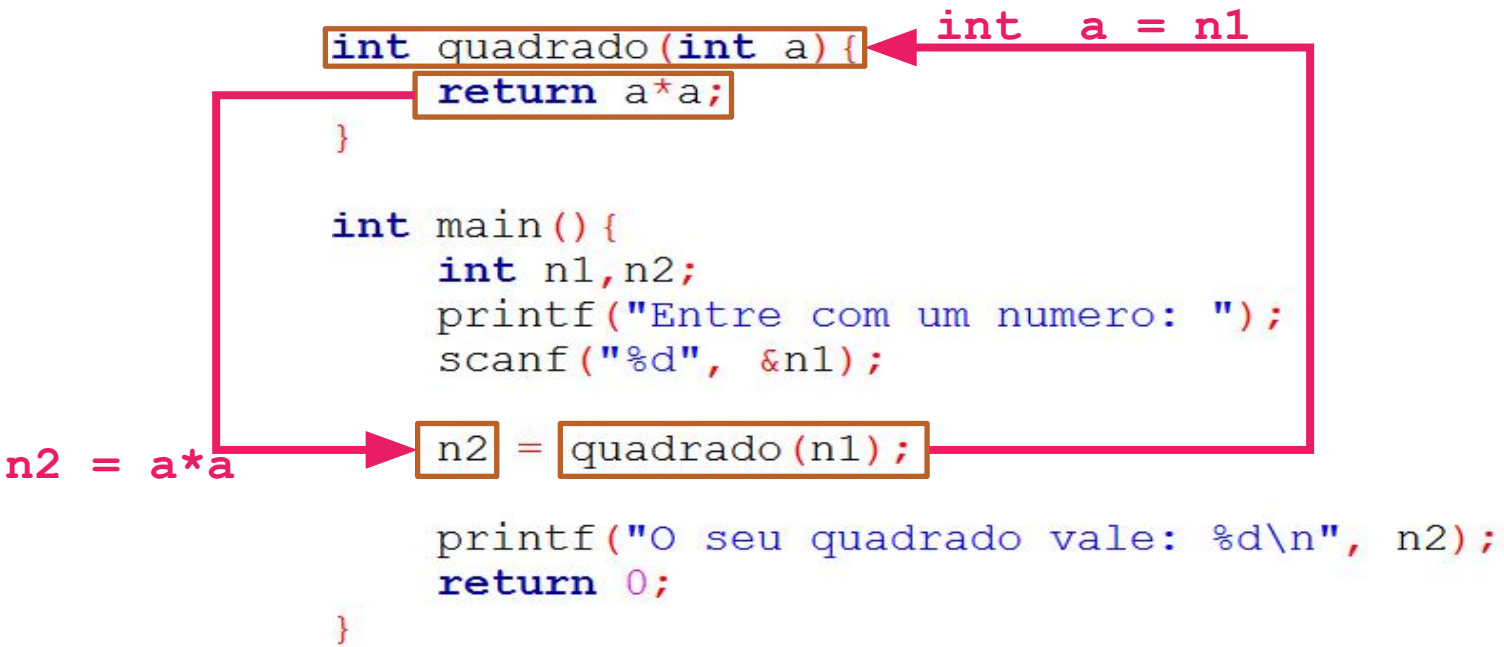
Função – Ordem de Execução

- Ao chamar uma função, o programa que a chamou é pausado até que a função termine a sua execução



Função – Ordem de Execução

- Ao chamar uma função, o programa que a chamou é pausado até que a função termine a sua execução



Declaração de Funções

- Funções devem ser declaradas antes de serem utilizadas, ou seja, antes da função **main**.
  - Uma função criada pelo programador pode utilizar qualquer outra função, inclusive as que foram criadas

```c
int quadrado(int a){
    return a*a;
}

int main(){
    int n1,n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);

    n2 = quadrado(n1);

    printf("O seu quadrado vale: %d\n", n2);
    return 0;
}
```

Declaração de Funções

- Podemos definir apenas o **protótipo da função** antes da função **main**.
  - O protótipo apenas indica a existência da função
  - Desse modo ela pode ser declarada após a função main().

```
tipo_retornado nome_função(parâmetros);
```

Os nomes dos parâmetros não são obrigatórios. Apenas os tipos. Protótipos equivalentes:

```
int soma(int a, int b);
int soma(int, int);
```

Declaração de Funções

- Exemplo de protótipo

```c
int quadrado(int a);

int main(){
    int n1,n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);

    n2 = quadrado(n1);

    printf("O seu quadrado vale: %d\n", n2);
    return 0;
}

int quadrado(int a){
    return a*a;
}
```

protótipo

Exercícios 1: conceitos básicos - AVA

Exercícios 2: prática de desenvolvimento - AVA

(30 minutos)

Escopo

- O escopo determina a validade de variáveis nas diversas partes do programa.
  - Variáveis locais
  - Variáveis globais
  - Parâmetros formais

Escopo

- Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas.
  - Um bloco começa quando abrimos uma chave e termina quando fechamos a chave.
  - Ex.: variáveis declaradas dentro da função.

```
int fatorial (int n){
    if (n == 0)
        return 1;
    else{
        int i;
        int f = 1;
        for(i = 1; i <= n; i++)
            f = f * i;
        return f;
    }
}
```

Escopo

- **Parâmetros formais** são declarados como sendo as entradas de uma função.
  - O **parâmetro formal é uma variável local** da função.
  - Ex.:

```
float quadrado(float x);
```

    - x é um parâmetro formal

Escopo

- Variáveis globais são declaradas fora de todas as funções do programa.
- Elas são conhecidas e podem ser alteradas por todas as funções do programa.
  - Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.

- ***Evite variáveis globais!***

Variáveis globais

Parâmetros formais são variáveis locais

Quais variáveis foram alteradas?

```c
#include <stdio.h>
int x,y;

int soma(int x, int y){
    x++;
    y--;
    printf("dentro x:%d y:%d soma:%d\n",x,y,x+y);
    return x+y;
}

int main()
{
    scanf("%d %d",&x,&y);
    printf("antes x:%d y:%d\n",x,y);
    printf("main x:%d y:%d soma:%d\n",x,y,soma(x,y));
    printf("depois x:%d y:%d\n",x,y);
    return 0;
}
```

Qual a saída?

```
Digite o valor de x: 14
Digite o valor de y: 18
Soma de 15 e 17: 32
Soma de 14 e 18: 32
x: 14
y: 18
```

Exercícios 3: escopo - AVA

Exercícios 4: prática de desenvolvimento - AVA

Referências

- BACKES, André. **Linguagem C:** completa e descomplicada. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3.Disponível na Biblioteca.