Algoritmos e Programação B

Modularização

Modularização

- Os algoritmos geralmente são desenvolvidos para solucionar problemas complexos.
- A solução destes problemas geram algoritmos extensos, que podem ser divididos em problemas menores.

- Organizar o algoritmo em pequenas partes destinados à solução de problemas menores.

Modularização

- As “pequenas” partes do algoritmo que contém uma solução parcial, além de favorecerem a melhor organização, proporcionam a **reutilização** de soluções.
- A criação de pequenas soluções parciais, gerando pequenos algoritmos, é chamada **Modularização**.

Exemplo 1

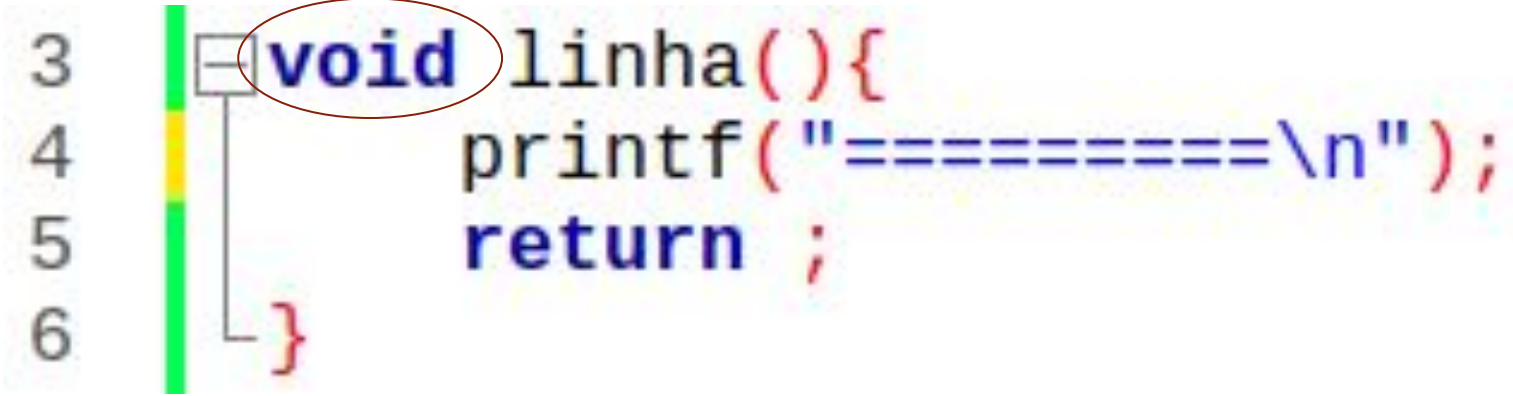
```c
#include <stdio.h>

void linha(){
    printf("============================================================\n");
    return ;
}

int main(){

    linha();
    printf("Exemplo do uso de Funções\n");
    linha();

    return 0;
}
```

Formato de uma função

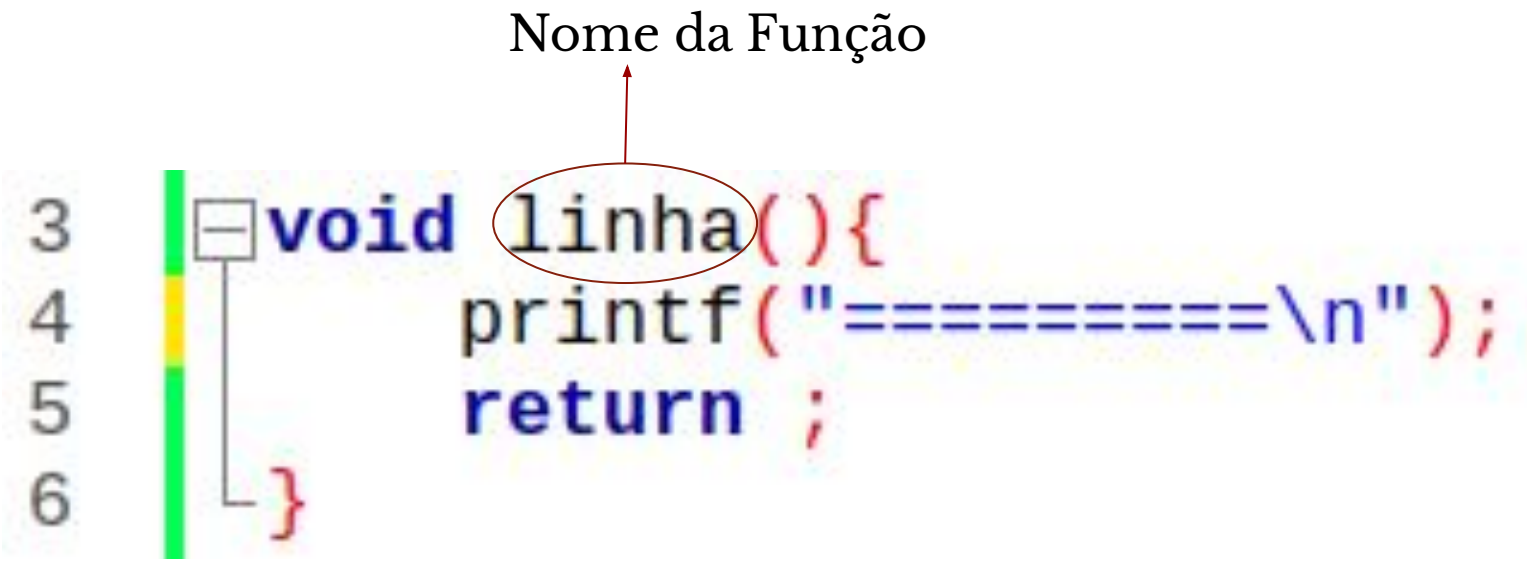
```
3   void linha(){
4       printf("==========\n");
5       return ;
6   }
```

Formato de uma função

Tipo do dado que será retornado pela função.

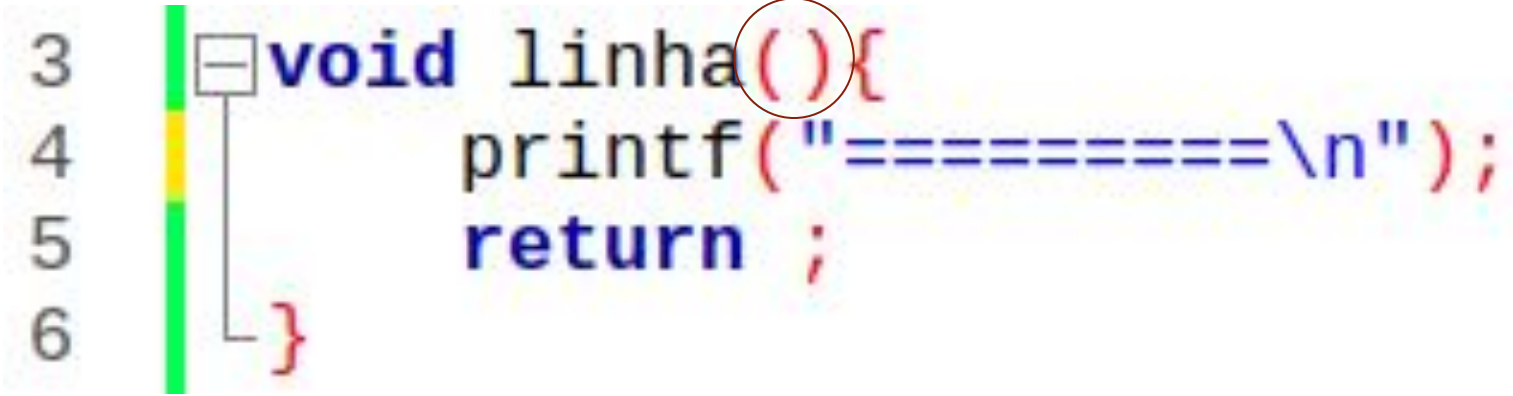


Formato de uma função



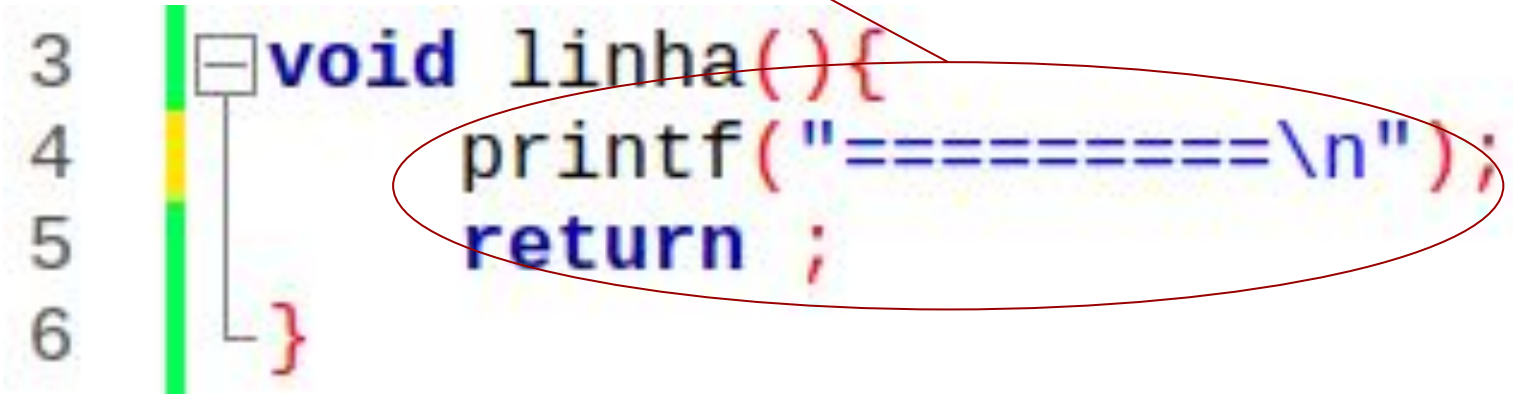
Formato de uma função

Espaço para declaração de parâmetros da função

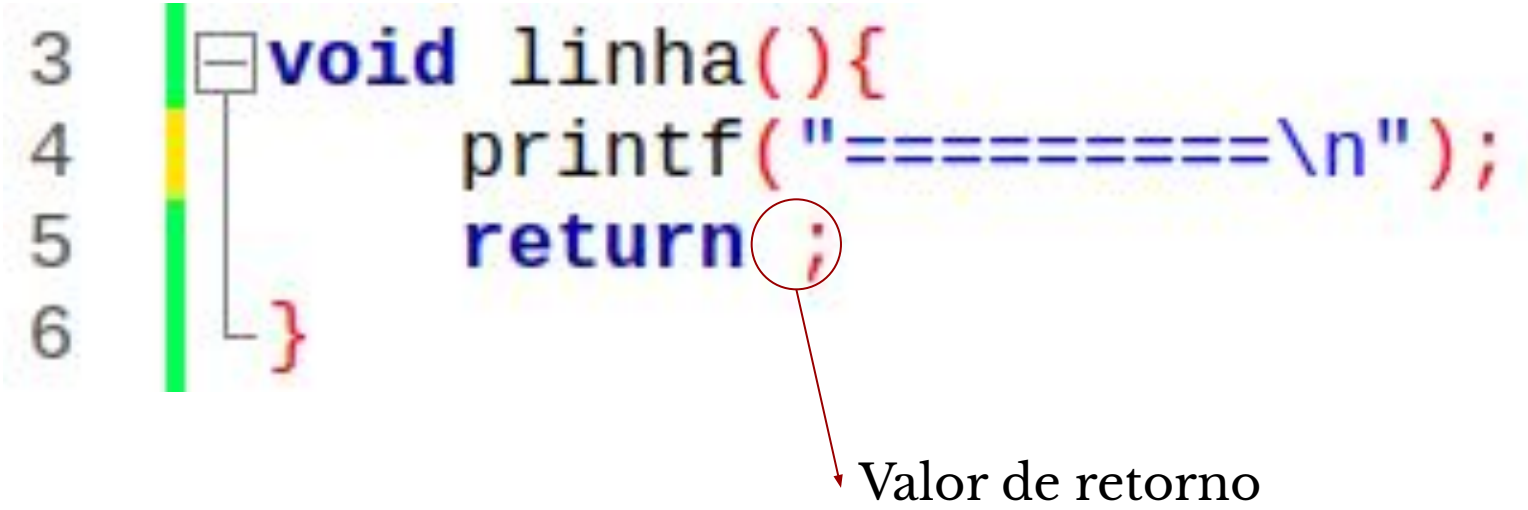


Formato de uma função

Corpo da função - contém o algoritmo.



Formato de uma função



Funções na Linguagem C

- Funções são os blocos de construção da linguagem C.
- Em uma função, temos a implementação de parte do algoritmo a solução de um problema.
- **Todo programa em C** é composto por pelo menos uma função: **main()**

Declaração de função

- Declarar uma função significa definir seu **nome**, especificar o **algoritmo** que ela implementa, o **tipo de retorno** do dado que ela devolve ao ser chamada e os **parâmetros** que a função pode receber.

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

Declaração de função

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

- **`tipo`** define o tipo de dado que o comando **`return`** da função irá devolver. O uso de **`void`** é indicado para funções que não retornam valores, pois impede o uso acidental destas em expressões.

Declaração de função

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

- **`nome_função`** identifica a função; é o nome da função e por este nome a função pode ser chamada em qualquer outra parte do programa.

Declaração de função

Lista de Parâmetros

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

- **`Lista de Parâmetros`** é uma lista que contém o **tipo de dado** e o **nome de um parâmetro** separados por vírgulas: **`tipo1 nome1, tipo2 nome2, ..., tipo3 nome3`**. Uma função pode não ter parâmetros, neste caso, apenas os parênteses **`()`** devem estar ao lado do nome da função.

Declaração de função

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

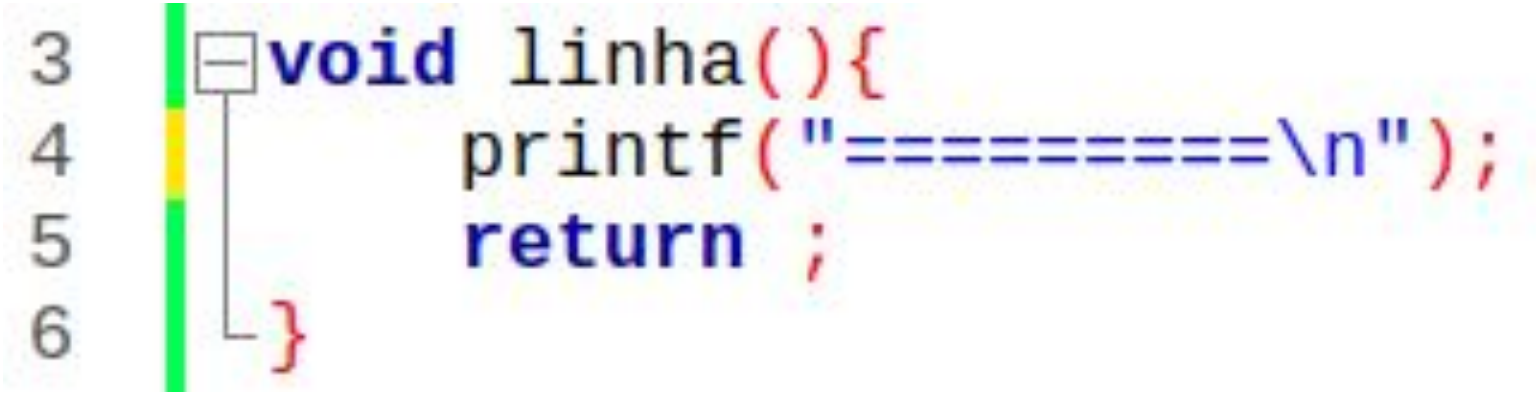
- **`corpo da função`** identifica os comandos a serem executados. É onde está implementado o algoritmo da função.

Declaração de função

```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //corpo da função, contém o algoritmo
    return ;
}
```

- **`return`** indica o dado que será retornado pela função, para a função chamadora. Pode ser retornado o conteúdo de uma variável ou diretamente um valor.

Declaração de função no Exemplo 1



Chamada de função

- Para uma função ser chamada, é necessário escrever o **nome da função** (sem o tipo de retorno) e os valores de argumentos, caso exista lista de parâmetros, em qualquer parte do programa.

Chamada de função no Exemplo 1

```
 8  int main(){
 9
10      linha();
11      printf("Exemplo do uso de Funções\n");
12      linha();
13
14      return 0;
15  }
```

- Nas linhas 10 e 11, temos a chamada da função `linha()`.
- Esta função:
  - têm lista de parâmetros?
  - retorna valor?

Exemplo 2

```c
#include <stdio.h>
#define pi 3.14159

void linha(){
    printf("================================================\n");
    return ;
}

float funcaoArea(float r){
    return pi * r * r;
}

int main(){
    float raio;
    float area;
    float perimetro;

    linha();
    printf("Cálculo da Área e do Perímetro de um Círculo\n");
    linha();
    printf("Insira o valor do raio do círculo: ");
    scanf("%f", &raio);
    area = funcaoArea(raio);
    printf("Area = %f\n", area);
    linha();

    return 0;
}

```

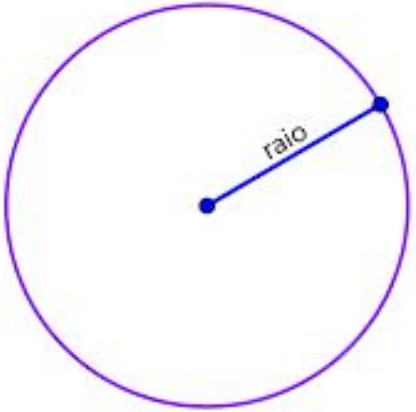
Exemplo 2

- Observe como acontece a sequência de execução das instruções do Exemplo 2.

```c
#include <stdio.h>
#define pi 3.14159

void linha(){
    printf("=====================================================\n");
    return ;
}

float funcaoArea(float r){
    return pi * r * r;
}

int main(){
    float raio;
    float area;
    float perimetro;

    linha();
    printf("Cálculo da Área e do Perímetro de um Círculo\n");
    linha();
    printf("Insira o valor do raio do círculo: ");
    scanf("%f", &raio);
    area = funcaoArea(raio);
    printf("Area = %f\n", area);
    linha();

    return 0;
}
```

Exemplo 2

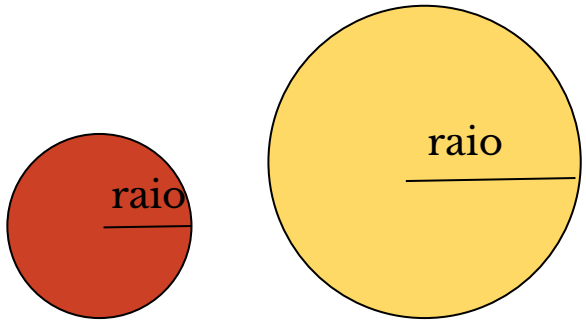
- Vamos implementar a função para calcular o perímetro do círculo!



$$A = \pi r^2 \quad p = 2\pi r$$

Exemplo 2

- E se houvessem dois círculos?
- Como poderemos calcular o raio e o perímetro de cada um?
- Vamos usar novamente as funções construídas!!



Para Construir a Modularização...

- Identificar o problema geral e quais tarefas ele deverá executar.
- Definir um programa principal para integrar todas as tarefas a serem executadas e, então, alcançar a solução principal.
- Elaborar o algoritmo e programar as soluções parciais para cada uma das tarefas identificadas

Para Construir a Modularização...

- ...utilizamos sub-rotinas, também conhecidas como **procedimentos** e **funções**:
  - Procedimentos são sub-rotinas que não retornam valores.
  - Funções são sub-rotinas que podem ou não retornar valores à rotina que as chamou.

A Linguagem C possibilita somente o uso de funções. Porém, podemos criar funções que não retornam valores, fazendo com que essas atuem como procedimentos. A sintaxe das funções na linguagem C é mantida.

Vamos praticar.... Exemplo 3

- Desenvolva um algoritmo, para ler o salário bruto (R$) de um funcionário e calcular:
  - o desconto (R$) do INSS;
  - a contribuição (R$) para o IRPF;
  - o valor do salário em dólares.
- Utilize funções para implementar o algoritmo e considere as seguintes tabelas do INSS e do IRPF:

Vamos praticar.... Exemplo 3

- INSS: considere que a Alíquota incide sobre o salário bruto.

Faixa salarial	Alíquota
Até R$ 1.302,00	7,5%
R$ 1.302,01 até R$ 2.571,29	9%
R$ 2.571,30 até R$ 3.856,94	12%
R$ 3.856,95 até R$ 7.507,49	14%

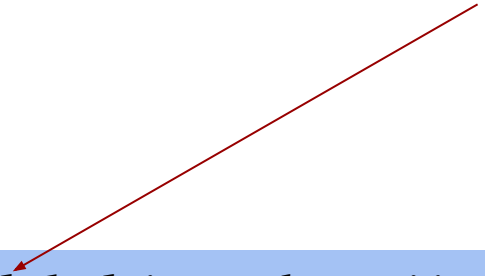
Vamos praticar.... Exemplo 3

- IRPF: considere que a Alíquota incide sobre o salário bruto.

Salário mensal	Alíquotas
Até R$ 2.112,00	Isento
De R$ 2.112,01 a R$ 2.826,65	7,5%
De R$ 2.826,66 a R$ 3.751,05	15%
De R$ 3.751,06 a R$ 4.664,68	22,5%
Acima de R$ 4.664,68	27,5%

Exemplo 4

- Ler um vetor de 10 elementos e mostrar todos os elementos pares digitados e sua posição no vetor.
  - Para desenvolver esta solução, vamos utilizar o que chamamos de **variável global**.



Variável Global é aquela variável criada fora de qualquer função.
Seu **escopo** (tempo de vida dentro do programa) é todo o programa, podendo ser acessada/alterada em qualquer função.

Exemplo 5

- Ler um número inteiro e usar uma função que retorna 0, se o número é par, ou 1, se o número é impar.