Programação Modularizada: FUNÇÕES

Introdução à Programação

Profª. Giorgia Mattos – giorgiamattos@gmail.com

Funções

- **Definição**
  - As funções, sub-rotinas ou ainda sub-programas, são blocos de instruções que realizam tarefas específicas. Seu código é carregado uma única vez e pode ser executado quantas vezes forem necessárias.
    - Exemplos de funções da linguagem C:
      - printf(), scanf(), pow(), sqrt(), strlen(), rand()...

Funções

- **Por que usar funções?**
  - Para permitir o reaproveitamento de código já construído;
  - Para evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa;
  - Para permitir a alteração de um trecho de código de uma forma mais rápida (alterar apenas **dentro** da função que se deseja);
  - Para dividir um problema em pequenas tarefas, os programas tendem a ficar menores e mais organizados;
  - Para facilitar a leitura do programa-fonte de uma forma mais fácil.

Funções

- Em geral os programas são executados linearmente, uma linha após a outra, até o fim do programa
- É possível realizar desvios na execução do programa quando são usadas funções. Os desvios acontecem quando uma função é chamada pela função main()

//Programa que recebe o valor atual do salário de um funcionário e calcula o seu novo salário.

```
int main () {
    float salario, reajuste, novo_salario;

    scanf ("%f", &salario);
    reajuste = ReajustaSalario (salario);
    novo_salario = salario + reajuste;
    printf ("Novo salario=%.2f",novo_salario);
    return 0;
}
```

```
float ReajustaSalario (float salarioAtual) {
    float valor;

    valor = salarioAtual*0.20;
    return valor;
}
```

A execução da função main() é desviada para a função ReajustaSalario

Funções

- Um programa em C pode ser composto de uma ou mais funções, sendo que a única obrigatória é a função main (), por onde começa a execução do programa
- Existem muitas funções predefinidas na linguagem C, ***sqrt()***, ***rand()*** etc, que são adicionadas aos programas pela diretiva #include, no momento da "linkedição"
- O programador pode criar várias funções, dependendo do problema a ser resolvido
- As funções às vezes precisam receber valores externos, chamados de parâmetros e também devolver algum valor produzido, denominado retorno
  - X = pow (2, 3);

- 2 e 3 são os parâmetros de entrada da função pow ()
- X contém o valor calculado por pow(), ou seja o retorno

Funções

- É recomendado que as funções não façam leitura de valores com **scanf**
    - Devemos passar os valores que elas precisam já prontos nos parâmetros
- É recomendado que as funções nunca exibam nada na tela com **printfs**
    - As funções só devem produzir os valores e a lógica da função **main**() que vai decidir se esses valores devem ser impressos ou não
- Essas recomendações podem ser quebradas caso você queira criar funções para entrada ou saída de dados
    - Mas lembre-se que o ideal é criar funções que possam ser reutilizadas

Funções

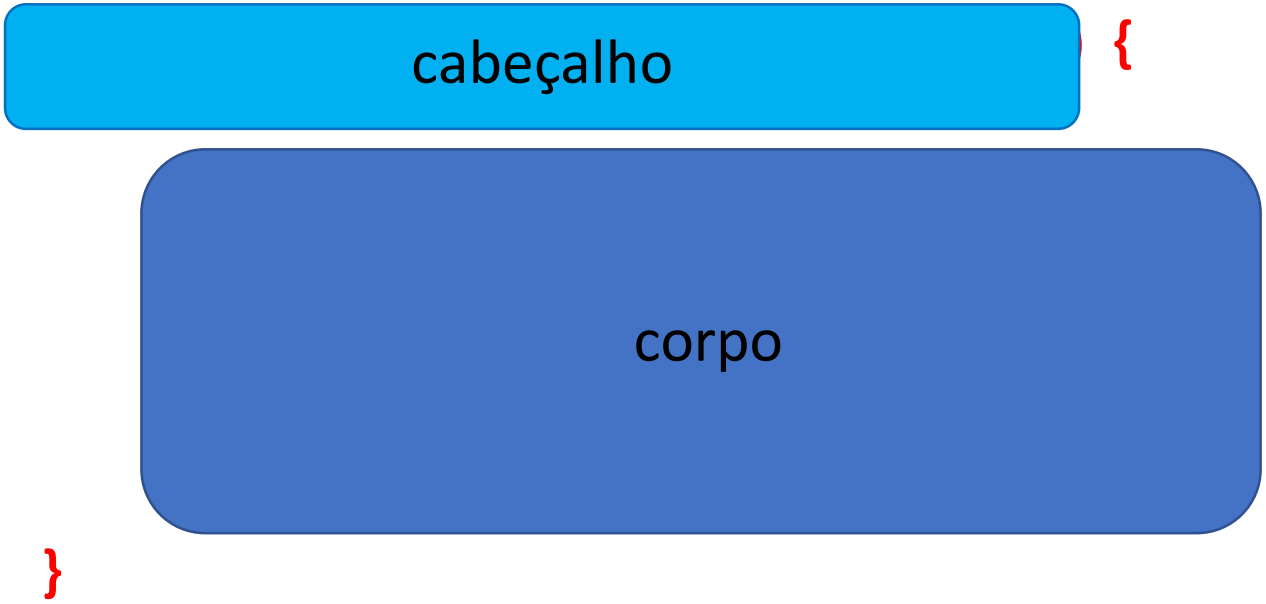
- As funções aparecem de três formas em um programa:
    - Definição
    - Chamadas e
    - Alusões

Funções

- A definição representa a implementação da função
  - Tem um cabeçalho com as informações:
    - Tipo do valor produzido
    - Nome da função e
    - Dados de entrada e saída (parâmetros)
  - Tem um corpo
    - Declarações e instruções que implementam a funcionalidade

Funções

- **Formato geral/sintaxe de uma Função:**



Funções

- **Formato geral de uma Função**

```
tipoFuncao  nomeFuncao (listaParametros) {
    // corpo da função
}
```

A **listaParametros** também chamada de **lista de argumentos**, é opcional, e é usada como variáveis locais à função

```
float CalculaSoma (float a, float b){
    float r;

    r = a + b;
    return r;
}
```

Funções

- **Elementos de uma função**
  - **Tipo da função**: o tipo da função pode ser qualquer um dos tipos definidos na linguagem e representa o tipo do dado que é retornado/devolvido pela função. Caso a função não retorne/devolva nenhum valor dizemos que ela é do tipo **void**. Caso não seja especificado nenhum tipo, por padrão a função retorna/devolve um inteiro. Mas é importante lembrar que sempre se deve declarar o tipo da função garantindo assim maior portabilidade.

  - **Lista de Parâmetros**: é constituída pelos nomes das variáveis que se deseja passar para a função separados por vírgulas e acompanhados de seus respectivos tipos. No caso da função não conter parâmetros a lista de parâmetros será vazia (void), mas mesmo assim será necessário utilizar os parênteses.

Funções

- **Elementos de uma função**
  - **Corpo da função:** contém as instruções/código em C, privativo da função, ou seja, nenhuma outra função poderá acessá-lo com nenhum comando, exceto por meio de uma chamada a função. Isso quer dizer que o código da função não pode afetar outras partes do programa, a menos que sejam utilizadas variáveis Globais. Isto porque as variáveis contidas em uma função são locais a ela, só existem naquela função. Essas variáveis são criadas para a função, no início da sua execução, e destruídas ao seu término.
- **O retorno da função**
  - Muitas vezes é necessário fazer com que uma função retorne/devolva um resultado. Podemos especificar um tipo de retorno indicando-o antes do nome da função. Mas para dizer a linguagem C o que vamos retornar precisamos da palavra reservada **return**.

Funções: Exemplos

```
float Media(float nota1, float nota2){
        return (nota1+nota2)/2;
}
```

```
void MostraMensagem(void){
                printf("Meu programa");
}
```

```
int EhPar(int n){
                if (n % 2 == 0)
                                return 1;
                else
                                return 0;
}
```

```
float Media(float nota1, float nota2){
    return (nota1+nota2)/2;
}
```

```
int EhPar(int n){
        if (n % 2 == 0)
                return 1;
        else
                return 0;
}
```

```
void MostraMensagem(void){
        printf("Meu programa");
}
```

Se a função não produz um valor, definimos o tipo de retorno como **void**

Se se ela não recebe nenhum parâmetro, podemos colocar um **void** entre os parênteses ou deixar vazio

Funções: Exemplos de utilização

```
float Media(float nota1, float nota2){
        return (nota1+nota2)/2;
}
```

```
void MostraMensagem(void){
                printf("Meu programa");
}
```

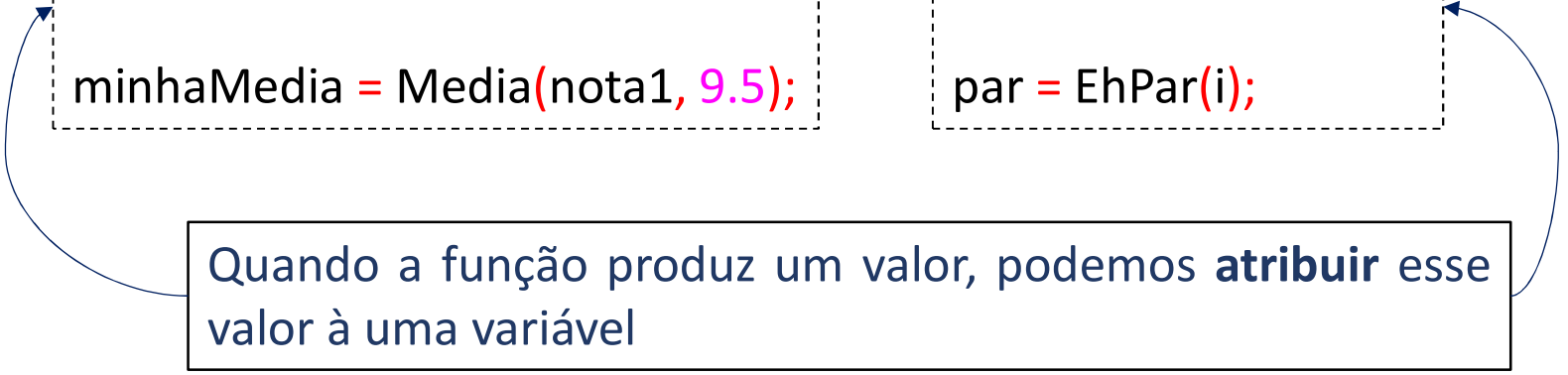
```
int EhPar(int n){
                if (n % 2 == 0)
                                return 1;
                else
                                return 0;
}
```

```
float minhaMedia, nota1 = 8.5;

minhaMedia = Media(nota1, 9.5);
```

```
int par, i = 3;

par = EhPar(i);
```



Quando a função produz um valor, podemos **atribuir** esse valor à uma variável

Funções: Exemplos de utilização

```
float Media(float nota1, float nota2){
        return (nota1+nota2)/2;
}
```

```
void MostraMensagem(void){
                printf("Meu programa");
}
```

```
int EhPar(int n){
                if (n % 2 == 0)
                                return 1;
                else
                                return 0;
}
```

```
float minhaMedia, nota1 = 8.5;

minhaMedia = Media(nota1, 9.5) + 0.5;
```

```
int par, i = 3;

if (EhPar(i))
                puts("i par");
```

Quando produzem valor, também podemos utilizar a função em **expressões** e **estruturas de controle**. O valor considerado é sempre o *valor produzido no momento da execução*.

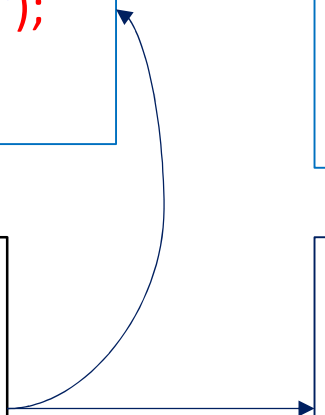
Funções: Exemplos de utilização

```
float Media(float nota1, float nota2){
        return (nota1+nota2)/2;
}
```

```
void MostraMensagem(void){
                printf("Meu programa");
}
```

```
int EhPar(int n){
                if (n % 2 == 0)
                                return 1;
                else
                                return 0;
}
```

Quando a função **não** produz valor, não podemos atribuir seu resultado a variáveis, nem usar em expressões ou estruturas de controle



```
MostraMensagem();
```

Quando a função não produz valor, só podemos chamá-la sozinha em uma linha de instrução.

Parâmetros das Funções

- Os parâmetros são semelhantes às variáveis
  - Estão associados a espaços de memória
  - Precisam ser declarados
  - São utilizados para auxiliar na execução das instruções
- Não são permitidas inicializações nem abreviações
- Se não possuir parâmetros, deixar em branco ou colocar **void** entre os parênteses

Chamadas de Funções

- Chamar uma função é transferir o fluxo de execução do programa para a função a fim de executá-la
- Ela pode ser chamada sozinha em uma instrução ou em uma expressão
  - Em expressões, sempre são avaliadas (chamadas) antes da aplicação de qualquer operador

Passagem de Parâmetros

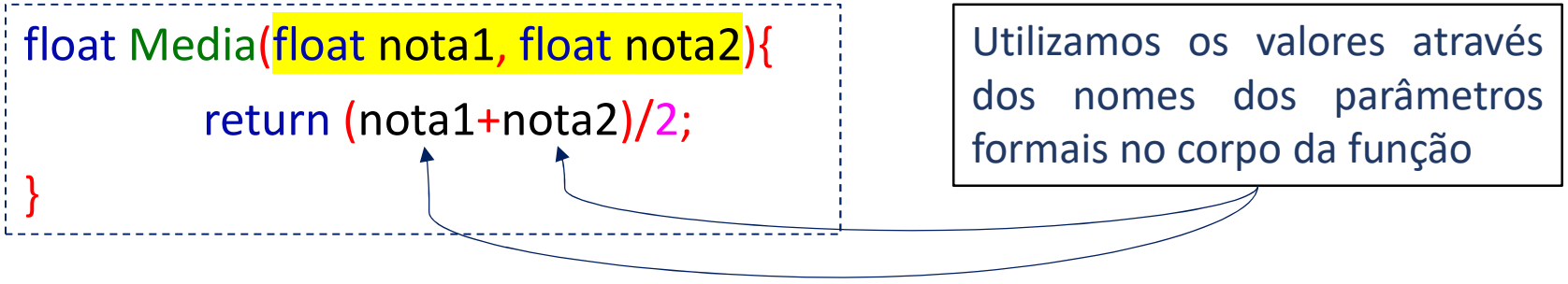
- Parâmetros formais:
  - Aqueles declarados na definição da função

- Parâmetros reais:
  - Aqueles passados numa chamada de função

Passagem de Parâmetros

- Parâmetros formais:
  - Aqueles declarados na definição da função

- Parâmetros reais:
  - Aqueles passados numa chamada de função



Passagem de Parâmetros

- Parâmetros formais:
  - Aqueles declarados na definição da função

- Parâmetros reais:
  - Aqueles passados numa chamada de função

```
float minhaMedia, nota1 = 8.5;

minhaMedia = Media(nota1, 9.5);
```

Passagem de Parâmetros

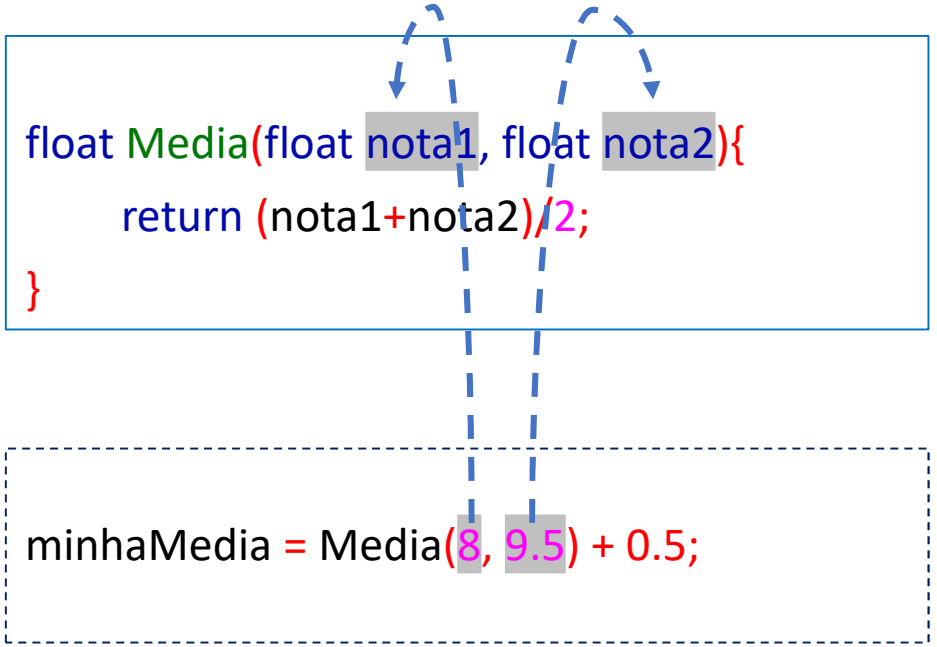
- Parâmetros formais:
  - Aqueles declarados na definição da função

- Parâmetros reais:
  - Aqueles passados numa chamada de função

- Numa chamada de função ocorre a ligação (ou casamento) dos parâmetros
  - De acordo com as regras de ligação
    - Mesmo número de parâmetros e
    - Tipos compatíveis

Passagem de Parâmetros

- Numa chamada de função ocorre a ligação (ou casamento) dos parâmetros
  - De acordo com as regras de ligação
    - Mesmo número de parâmetros e
    - Tipos compatíveis



Tipos de Passagem de Parâmetros

- Passagem por valor:
  - O parâmetro formal recebe uma copia do parâmetro real
  - Qualquer modificação dentro da função só altera o parâmetro formal

- Passagem por referência:
  - O parâmetro real é uma variável
  - O parâmetro real se torna o parâmetro formal (mesmo espaço de memória)
  - Modificações são feitas também no parâmetro real
  - Não existe em C!!

Modos de Parâmetros

- Os parâmetros são a forma que as partes do programas tem para se comunicar com as funções

- **Parâmetros de entrada** informam a função os dados que ela deve utilizar para executar sua funcionalidade

Instrução return

- Encerra imediatamente a execução da função
- Em funções do tipo **void**:
  - O uso é opcional, caso não tenha a função retorna após a ultima instrução
  - Pode ter mais de uma, cada uma para diversos pontos de encerramento da função
  - Não especificam um valor

```
void DizOi(void){
        printf("Oi");
}
```

```
void TalvezDigaOi(int sim){
        if (sim == 1){
                puts("oi");
                return;
        }else{
                return;
        }
}
```

```
void DizOla(){
        printf("Ola");
        return;
}
```

Instrução return

- Encerra imediatamente a execução da função
- Em funções de tipo **diferente de void**:
  - São acompanhadas de um valor ou expressão especificando o retorno
  - **Devem** ter pelo menos uma instrução retornando um valor compatível com o tipo de retorno da função – pode ocorrer *conversão implícita*
  - Podem ter mais de uma, cada uma para diversos pontos de encerramento da função
    - Isso **não** significa que a função pode retornar mais de um valor por vez

Instrução return

- Em funções de tipo **diferente de void**:
  - São acompanhadas de um valor ou expressão especificando o retorno
  - **Devem** ter pelo menos uma instrução retornando um valor compatível com o tipo de retorno da função – pode ocorrer *conversão implícita*
  - Podem ter mais de uma, cada uma para diversos pontos de encerramento da função
    - Isso **não** significa que a função pode retornar mais de um valor por vez

```
int QuarentaEDois(){
        return 42.0;
}
```

```
int SomaAteN(int n){
        int i, soma = 0;
        for(i = 0; i < n; i++)
                soma += i;
        return soma+n;
}
```

```
int EhPar(int n){
        if (n % 2 == 0)
                return 1;
        else
                return 0;
}
```

Alusões de Funções

- **Localização das funções dentro do código fonte**
  - Antes que a função seja usada pela primeira vez é necessário que ela seja declarada. Isso ocorre porque o compilador C (em sua grande maioria), por padrão, assume que uma função devolve um valor inteiro. Portanto se alguma função devolver um tipo diferente de inteiro o compilador gera um código errado para chamá-la
  - Por isso é importante declarar o tipo que a função retorna no inicio do programa e isso pode ser feito utilizando os **protótipos**
  - Através do protótipo usado no inicio do programa é possível que o compilador verifique se existem erros nos tipos de dados entre os argumentos usados para chamar uma função e a definição de seus parâmetros. Além de verificar se a quantidade de argumentos é igual a quantidade de parâmetros, caso contrário causará erros na execução do programa

Alusões de Funções

- **Sintaxe do protótipo**

  tipo NomeFuncao (tipo parametro1, tipo parametro2, ...);

✓ Caso a função não utilize parâmetros pode-se utilizar o **void**

tipo NomeFuncao (**void**);

Atenção!!

Os protótipos são utilizados quando as funções estão definidas **após a função main()**. Se as funções estiverem antes da main () não há a necessidade de declará-los.

```
/*Este programa calcula o Fatorial de um número informado pelo usuário */
#include <stdio.h>
int CalculaFatorial (int x);    //Protótipo da função

int main () {          //Função obrigatória em um programa C
      int N, fat;      //Variáveis locais

      printf ("Digite um numero para calcular o fatorial : ");
      scanf ("%d", &N);
      fat = CalculaFatorial (N);
      printf ("\n\nFATORIAL = %d\n", fat);
      return 0;
}
```

/* Esta função recebe como parâmetro o número digitado pelo usuário como entrada do programa. O valor de N é copiado para o parâmetro x da função. A função calcula o fatorial de x e retorna o fatorial deste número */

```
int CalculaFatorial (int x){
   int c, f=1;    //Variáveis locais, existem enquanto a função está sendo executada
   for (c=1; c<=x; c++)
      f = f*c;      //f guarda o fatorial do x

   return (f);    //Retorna o valor calculado, atribuído a variável f
}
```

Funções – Exercícios práticos

1. Faça um programa que mostre se um número é par ou ímpar utilizando uma função que retorne 0 se o número for par ou 1 se o número for ímpar.
2. Escreva um programa que leia dois números e utilize uma função que retorne o maior valor. Mostre o resultado.
3. Crie um programa contendo uma função que receba 3 números e retorne o maior valor. Atenção: use a função escrita da questão 2.
4. Faça uma função que desenhe linhas de caracteres na tela. A função receberá como argumentos o tipo de caractere e o número de linhas que deverá desenhar.
5. Faça um programa contendo uma função que receba 3 inteiros *a, b, c*, sendo *a* maior que 1. A função deverá somar todos os inteiros entre *b* e *c* que sejam divisíveis por *a* (inclusive *b* e *c*) e retornar o resultado para ser mostrado na função main().

Funções – Exercícios práticos

6. Faça uma função que receba como argumento os valores dos lados de um triângulo. A função deverá retornar 0 se o triângulo for equilátero (os 3 lados iguais), 1 se for isósceles (2 lados iguais) ou 2 se for escaleno (os 3 lados diferentes).
7. Faça uma função que recebe a altura e o sexo de uma pessoa por parâmetro e retorna o seu peso ideal. Calcular o peso ideal usando a fórmula
   peso = 72.7 * altura - 58 (para homens)
   peso = 62.1 * altura - 44.7 (para mulheres).

8. Escreva uma função que recebe por parâmetro um valor inteiro e positivo N e retorna o valor de S.

$$S = 1 + 1/1! + 1/2! + 1/3! + 1 /N!$$

- Use a função fatorial já apresentada para determinar o valor de S
- Escreva uma função que mostre a sequencia S como formatada acima.