



# TÉCNICAS DE PROGRAMAÇÃO

Aula 05 – Conceitos e Características de uma Função



# Objetivos de Aprendizagem

1. Identificar os princípios básicos para a construção de programas usando funções;
2. Desenvolver programas tradicionais usando funções;
3. Identificar os princípios básicos para a construção de programas usando funções.
4. Desenvolver programas tradicionais usando funções com e sem retorno de um valor.




# Prova 1 (A1 – Unidade 1 – N1)

- Lista de Revisão: entrega até dia **15/10**:
  - Valor de até 4,0 (quatro) pontos
  - Grupos de 6 (seis) integrantes
    - Enviar nomes até **10/10** para **o(a) embaixador(a) da turma**
  - **Enviar o arquivo zip com o código fonte** para o embaixador
- Revisão dia **17/10** (sábado) das 8h às 9h30
  - Resolução de questões da lista de Revisão
- Prova dia **22/10**
  - Todo o conteúdo visto até dia **08/10/2020**
  - Valor: 6,0 (seis) pontos
  - Individual
  - Questões abertas e de múltipla escolha



# Modularização

- A linguagem C, como qualquer linguagem, permite dividir um programa grande em pedaços pequenos e simples, ou módulos (tradicionalmente chamadas de funções).
  - Essa técnica é chamada de dividir para conquistar.
- 



# Modularização

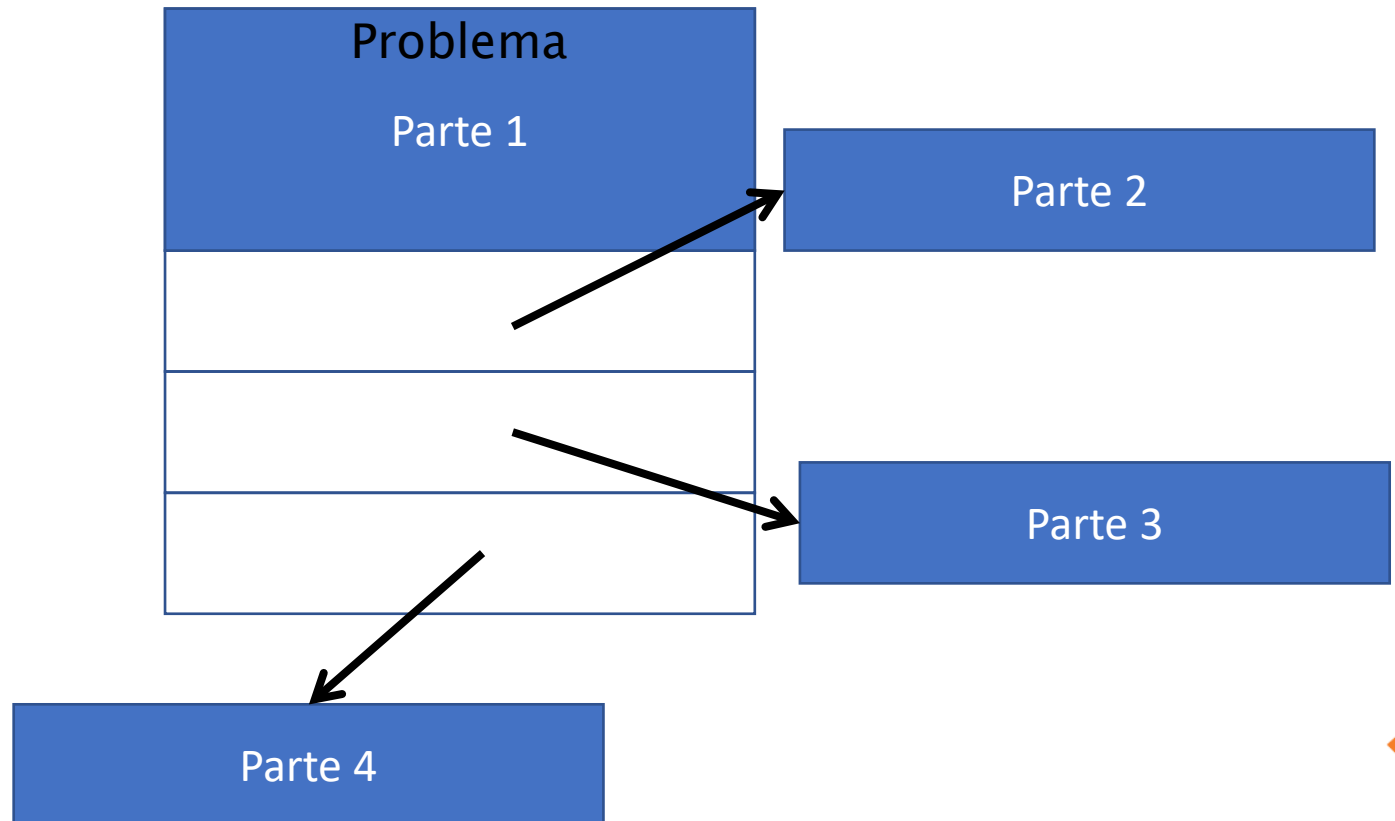
- Situação



Problema

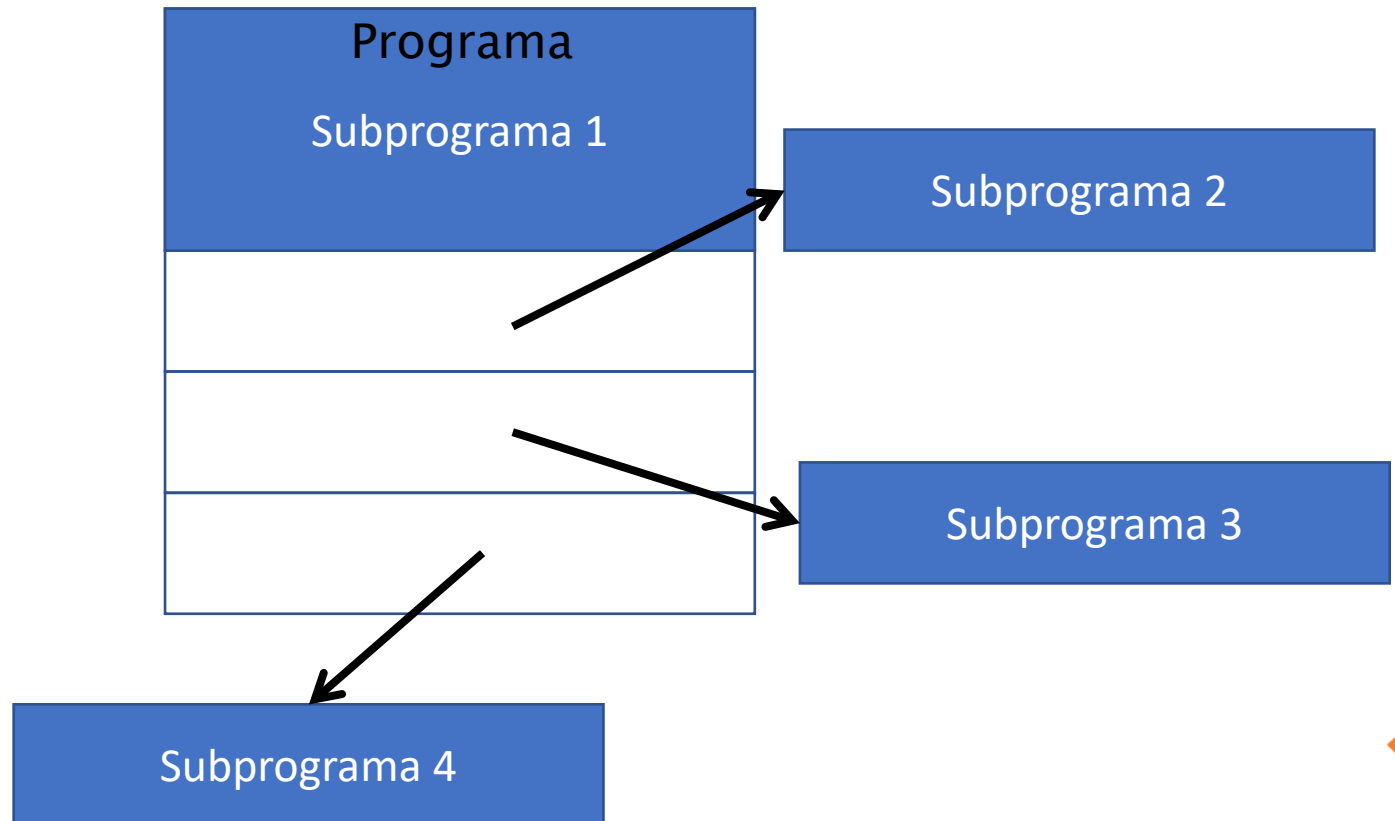
# Modularização


- Solução




# Modularização: subprograma

- Solução





# Modularização: subprograma

- A solução do problema original é implementada através da composição dos subprogramas.
  - Subprogramas podem ser vistos como blocos de construção com os quais montamos um programa.
  - Esta técnica de construção de programas é chamada de **MODULARIZAÇÃO**.
- 



# Modularização: Bloco

- Um bloco consiste em um conjunto de declarações e um conjunto de comandos delimitados por indicadores de início e fim de bloco.
- Em C, os delimitadores de bloco são { e }.

```
{  
    int a, b, c;  ← Declarações de variáveis  
    a = 100;  
    b = 200;  
    c = a + b;  
    printf("%d", c);  
}
```

Comandos

# Modularização: subprograma


- Um subprograma é um bloco de programa constituído por um conjunto de declarações e por um conjunto de comandos e identificado por um nome.
- Exemplo:

```
void adicao()  
{  
    int a, b, c; ← Declarações de variáveis  
    a = 100;  
    b = 200;  
    c = a + b;  
    printf("%d", c);  
}
```

Comandos



# Modularização

- **Observação:** Embora os programas **não modularizados** possam resolver a maioria dos problemas, fazer a **modularização** facilita a vida do programador.
  - Um programa pode conter uma ou mais funções, sendo que uma delas deve ser ***main()***.
- 



# Modularização


- Observação: A execução do programa sempre começa na função ***main()***, e quando o controle do programa encontra uma instrução que inclui o nome de uma função, a função é chamada.
- Já escrevemos programas que chamam função. Como exemplo:

```
printf("Meu primeiro programa");  
// Chamando a função printf
```







# Porque Usar Modularização

- Para permitir o reaproveitamento de código já construído;
  - Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
  - Para permitir a alteração de um trecho de código de uma forma mais rápida;
- 



# Porque Usar Modularização


- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
  - Para facilitar a leitura do programa-fonte de uma forma mais fácil;
  - Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.
- 



# Modularização: subprograma

- Existem dois tipos de subprograma:
  - Procedimentos
    - Executam tarefas
  - Funções
    - Calculam Valores





# Modularização: subprograma

- Funções e Procedimentos
  - Toda função ou procedimento deve ser pelo menos declarada antes da função principal main().
    - Declara a função/procedimento antes da main() e a implementa depois.
    - Implementa a função/procedimento diretamente antes da main().





# Modularização: subprograma

- Funções e Procedimentos
  - Exemplo: Procedimento que não recebe parâmetros e não retorna valor.

```
#include <stdio.h>
```

```
void desenha();
```

```
main()
```

```
{
```

```
    printf("Usando procedimento.");  
    desenha( );
```

```
}
```

Assinatura ou  
protótipo do  
procedimento

Implementação do  
procedimento

```
void desenha()
```


```
{
```

```
    int i;  
    for (i = 0; i <= 10; i++)  
        printf("--\n");
```

```
}
```




# Protótipo de Procedimento/Função

- Uma função não pode ser chamada sem antes ter sido declarada.
  - A declaração de uma função é dita protótipo da função e é uma instrução, em geral colocada no início do programa, que estabelece o tipo da função e os argumentos que ela recebe.
- 



# Protótipo de Procedimento/Função

- O protótipo da função permite que o compilador verifique a sintaxe de sua chamada.
  - **Observação:** O protótipo de uma função deve preceder sua definição e sua chamada.
  - O protótipo de uma função tem a mesma forma da primeira linha da definição da função, exceto por terminar com ponto-e-virgula após o fechamento do parêntese.
- 



# Exemplo

- Suponha que queiramos escrever um *programa* para calcular a média de um determinado aluno. Para isso, vamos escrever uma função que calcule a média.
- **Protótipo de função**  
`float media (float nota1, float nota2);`



# Procedimento

- Sintaxe:

```
void nome_procedimento(lista_parâmetros)
{
    declarações_de_variáveis_do_procedimento;
    lista_de_comandos_do_procedimento;
}
```





# Procedimento


- Exemplo:
  - Exibir na tela uma linha com 20 asteriscos.

```
void linha(){  
    int i;  
    for (i = 1; i <= 20; i++)  
        printf('*');  
    printf('\n');  
}
```





# Procedimento

- A definição de um procedimento associa um nome a um bloco de declarações e comandos.
  - Os comandos que compõem o bloco do procedimento têm a “missão” de executar uma determinada tarefa para o programa.
- 



# Procedimento

► Exemplo de chamada a um procedimento

```
void main(){  
    int i;
```

```
    linha(); /* escreve uma linha de asteriscos */  
    puts("Numeros entre 1 e 5");
```

```
    linha(); /* escreve outra linha de asteriscos */
```

```
    for(i = 1; i <= 5; i++)  
        printf("%d\n", i);
```

```
    linha(); /* escreve outra linha de asteriscos */
```


```
}
```







# Procedimento

- Chamada (Execução) de um Procedimento
    - Uma referência a um nome de procedimento dentro de um programa provoca a execução do bloco de comandos do procedimento.
    - Ao término da execução do procedimento, a execução do programa continua a partir do ponto onde este foi chamado.
- 

# Procedimento

- Chamada (Execução) de um Procedimento


```
void main(){
```

- `int i;`
- `linha();`
- `puts("Numeros entre 1 e 5");`
- `linha();`
- - `for(i = 1; i <= 5; i++)`
    - `printf("%d\n", i);`
    - `linha();`

- `void linha(){`
- `int i;`
- `for (i = 1; i <= 20; i++)`
  - `printf('*');`
  - `printf('\n');`
- `}`



# Exercício - Procedimentos

1. Escreva um programa que receba o nome e o sobrenome do usuário (armazene em strings).
    - Crie um procedimento chamado **void imprimir()** que imprime o nome e sobrenome do usuário.
  2. Escreva um procedimento **void menu()** que represente o menu de uma calculadora mostrando ao usuário as opções a serem escolhidas.
- 



# Função: estrutura

- A estrutura de uma função em C é semelhante à da função ***main***. A diferença é que a função ***main*** possui um **nome especial**.
- O código que descreve o que a função faz é chamado de ***definição da função***. Sua forma geral é a seguinte:

```
<tipo> <identificador> (<parâmetros>) {  
    return <valor>;  
}
```





# Função


## ► Sintaxe:

```
tipo_retorno nome_função (lista_parâmetros)
{
    declarações_de_variáveis_da_função;
    lista_de_comandos_da_função;
}
```

## ► Exemplo:


- Calcular o valor da soma de dois números reais.

```
float soma(float a, float b) {
    float aux;
    aux = a + b;
    return aux;
}
```





# Tipos de Função

- Os tipos de uma função é definido pelo valor que ela retorna por meio do comando ***return***. Uma função é do tipo ***int*** quando retorna um valor do tipo ***int***.
  - Os tipos de funções C são os mesmos de variáveis, exceto quando a função não retorna nada. Nesse caso, ela é do tipo ***void***.
- 



# Definição de Função


- **Definição da Função**

```
float media(float nota1, float nota2) {  
    float med;  
    med = (nota1 + nota2)/2;  
    return med;  
}
```





# Definição de Função


- O cabeçalho da função: ***float media (float nota1, float nota2)*** inicia com o tipo de retorno (***float***).
  - Depois vem o nome da função e finalmente os tipos e nomes dos argumentos (parâmetros).
- 





# Exemplo

```
#include<stdio.h>
#include<stdlib.h>
// Protótipo da função media
float media (float nota1, float nota2);
int main() {
    float n1, n2;
    scanf("%f", &n1);
    scanf("%f", &n2);
    // Imprime a média do aluno na saída padrao
    printf("A média do aluno é: %f\n", media(n1,n2));
    system("PAUSE");
    return 0;
}
```





# Exemplo

```
// Definição da função media
float media (float nota1, float nota2) {
    float med;
    med = (nota1 + nota2)/2;
    return med;
}
```






# Função: chamada


```
float soma(float a, float b) {  
    float aux;  
    aux = a + b;  
    return aux;  
}
```

```
void main () {  
    float x,y,z;  
    printf("Digite X:");  
    scanf ("%f", &x);  
    printf("Digite Y:");  
    scanf ("%f", &y);  
    z = soma(x,y);  
    printf("Soma: %f",z);  
}
```





# Parâmetros da Função

- Parâmetros são valores passados para as funções para que as mesmas possam realizar a sua lógica dependendo do valor dos mesmos.
  - Existem dois tipos de passagem de parâmetros: **passagem por valor** **passagem por referência**.
- 

# Parâmetros da Função


- As variáveis que receberão as informações enviadas a uma função são chamadas de parâmetros (argumentos).
- A função deve declarar essas variáveis entre parênteses, no cabeçalho de sua definição.

Parâmetros

`float media(float nota1, float nota2)`




# Função

- Definição:
    - A definição de uma função associa um nome a um bloco de declarações e comandos.
    - Os comandos que compõem o bloco da função têm a “missão” de calcular um valor que deve ser informado pela função.
- 




# Função

- Chamada (execução) de Função:
    - Uma referência a um nome de função dentro de um programa provoca a execução do bloco de comandos da função.
    - Ao término da execução da função, a execução do programa continua a partir do ponto onde a função foi chamada.
- 




# Observações


- O protótipo de uma função deve preceder sua definição e sua chamada.
  - As funções definidas antes de serem chamadas não necessitam de protótipo.
  - O tipo de uma função é determinado pelo valor que ela retorna via comando ***return***, e não pelo tipo de argumento que ela recebe.
- 





# Observações

- Uma função pode não retornar nenhum tipo, essa função é do tipo ***void*** e não precisar o comando ***return***.
  - O comando ***return*** termina a execução da função e retorna o controle para a instrução seguinte do código de chamada.
  - O comando ***return*** pode retornar somente um único valor para a função que chama.
- 



# Exercício - Funções

- Modifique o Exercício 2 para criar uma calculadora.
    - O programa deve executar **menu()** e mostrar as opções ao usuário.
    - O programa deve pedir para o usuário entrar com dois números reais.
    - Por fim, o programa deve chamar a função escolhida pelo usuário:
    - **float somar(float a, float b), float subtrair(float a, float b), float multiplicar(float a, float b) ou float dividir(float a, float b).**
    - O retorno da função deve ser impresso na função **main()**.
- 