

Fundação Hermínio Ometto

Bacharelado em Sistemas de Informação

SIF009 - Linguagem de Programação I

Prof. Dr. Sérgio Luis Antonello

Aula 10

06/10/2020

Plano de ensino

1. Unidade I - Programação estruturada e Linguagem C (objetivos b, c e d).

- 1.1. Conceitos de programação estruturada.
- 1.2. Estrutura de um programa de computador.
- 1.3. Códigos fonte, objeto e executável.
- 1.4. Biblioteca de códigos.
- 1.5. Compiladores e Interpretadores.
- 1.6. Processos de compilação e link edição.
- 1.7. Identificação dos tipos de erros e alertas (léxicos, sintáticos e semânticos).
- 1.8. Depuração de código.
- 1.9. Palavras reservadas.
- 1.10. Tipos de dados.
- 1.11. Constantes. Variáveis simples e estruturadas. Escopo de variáveis.
- 1.12. Operadores e precedência.
- 1.13. Expressões aritméticas, lógicas e relacionais.
- 1.14. Comandos.
- 1.15. Ambientes de desenvolvimento e programação.

2. Unidade II - Estruturas de controle (sequência, decisão e repetição), registro e arquivo (objetivos a, c, d).

- 2.1. Comandos if e switch.
- 2.2. Comandos for, while e do while.
- 2.3. Blocos de comandos e aninhamento.
- 2.4. Definição de tipos.
- 2.5. Registro.
- 2.6. Arquivo: leitura e gravação de dados em disco.

3. Unidade III - Ponteiros e Funções (objetivos a, c, d, e).

- 3.1. Ponteiros.
- 3.2. Funções.
- 3.3. Passagem de parâmetro por valor.
- 3.4. Passagem de parâmetro por referência.

4. Unidade IV- Strings e Variáveis indexadas (objetivos a, c, d).

- 4.1. Manipulação de strings.
- 4.2. Manipulação de caracteres.
- 4.3. Declaração e manipulação de vetores.
- 4.4. Declaração e manipulação de matrizes.

Plano de ensino

Data	Atividade
04/08	Aula 01
11/08	Aula 02
18/08	Aula 03
25/08	Aula 04
01/09	Aula 05
08/09	Aula 06
15/09	Prova 1
22/09	Aula 08
29/09	Aula 09
06/10	Aula 10

Data	Atividade
13/10	Aula 11
20/10	Maratona FHO de Programação
27/10	Aula 13
03/11	Aula 14
10/11	Aula 15
17/11	Prova 2 Entrega Trabalho
24/11	Prova SUB
01/12	Semana Científica
08/12	Aula 19
15/12	Aula 20

Sumário da aula

Primeiro momento (revisão)

- ✓ Funções
- ✓ Escopo de variáveis
- ✓ Retorno de valores
- ✓ Passagem de parâmetro por valor

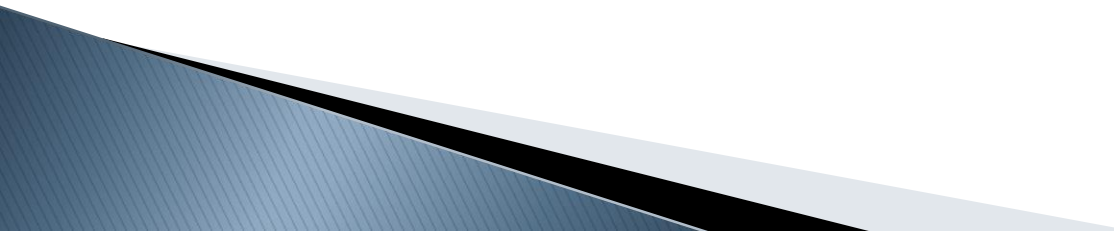
Segundo momento (conteúdo)

- ✓ Ponteiro
- ✓ Manipulação de endereços de memória por meio de ponteiro
- ✓ Passagem de parâmetro por referência

Terceiro momento (síntese)

- ✓ Retome pontos importantes da aula

1. Primeiro momento: revisão

- ▶ O que é uma função?
 - ▶ Ao ser chamada (evocada) uma função pode receber dados?
 - ▶ Como chamamos esses dados?
 - ▶ De que tipo são esses dados?
 - ▶ Uma função pode retornar dados?
 - ▶ Quantos?
 - ▶ De que tipo?
 - ▶ Se a função não retornar nenhum dado, o que deve ser feito na declaração?
- 

1. Primeiro momento: revisão

Exemplo

```
int soma(int n1,int n2) {  
    int resultado;  
    resultado=n1+n2;  
    return resultado;  
}
```

Parâmetros n1 e n2:
correspondem aos
valores recebidos de
main().

resultado: valor a retornar
para quem chamou soma().

```
int main() {  
    int valor1, valor2, valsoma;  
    printf("\nEntre com dois números inteiros\n");  
    scanf("%d",&valor1);  
    scanf("%d",&valor2);  
    valsoma = soma(valor1, valor2);  
    printf("\n%d + %d = %d\n", valor1, valor2, valsoma);  
}
```

Argumentos valor1 e
valor2: correspondem
aos valores a serem
enviados para soma().

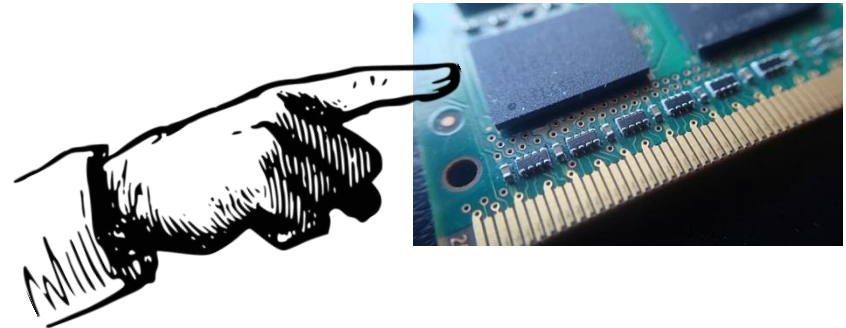
Valor retornado de
soma().

1. Primeiro momento: revisão

- ▶ Correção de exercícios

2. Segundo momento

- ▶ Ponteiro
- ▶ Manipulação de endereços de memória por meio de ponteiro
- ▶ Passagem de parâmetro por referência




3. Ponteiro

Todo dado a ser manipulado em um programa está armazenado na memória do computador, por um certo conjunto de bytes.

Cada um destes conjuntos de bytes tem um nome e um endereço de localização na memória.

Um ponteiro é um recurso de algumas linguagens de programação que permitem acesso e manipulação dos dados diretamente na memória do computador, por meio de seus endereços de memória.

Em um ponteiro armazena-se o endereço de memória e por meio dele, o dado referente a esse endereço pode ser acessado e manipulado.



3. Ponteiro

- É utilizado para armazenar um endereço de memória.
- O ponteiro deve ser do mesmo tipo do conteúdo armazenado neste endereço.
- O que caracteriza a declaração de um ponteiro é o uso de um asterisco (*) na frente do nome do mesmo.

Exemplos de declaração de ponteiros:

```
int *p;
```

```
float *x;
```

```
char *nome;
```

3. Ponteiro

- Para atribuir um valor a um ponteiro deve-se copiar para ele um endereço de memória.
- Se o endereço desejado for de uma variável declarada no programa, basta utilizar o & (e-comercial) antes do nome da variável, que seu endereço estará sendo utilizado.

Exemplo:

```
int var;  
int *p;  
var = 10;  
p = &var;
```

Declaração do ponteiro de nome "p"


O ponteiro "p" recebe o endereço de memória da variável "var", ou seja, agora o ponteiro "p" está apontando para "var".

3. Ponteiro: operadores

Operador * (asterisco):

- na declaração, informa que uma variável irá armazenar o endereço de outra variável; ou:
- informa ao computador que você deseja o conteúdo que está no endereço (pode ser lido como “no endereço”).

Exemplo: `q = *m;`



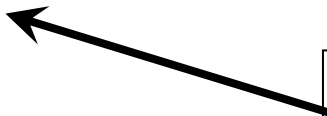
q recebe o conteúdo armazenado no endereço de memória apontado pelo ponteiro m.

Operador & (e-comercial):

- retorna o endereço de uma variável;
- pode ser lido como “o endereço de”.

O ponteiro “p” recebe o endereço de memória da variável “var”, ou seja, agora o ponteiro “p” está apontando para “var”.

Exemplo: `m = &count;`



O ponteiro m recebe o endereço de memória da variável count.

4. Aritmética de ponteiros

- Operações com ponteiros para que ocorra deslocamentos na memória. Esse processo ocorre a partir do endereço de memória armazenado no ponteiro.
- O deslocamento (quantidade de bytes na memória) é proporcional ao tipo que o ponteiro foi declarado.

Exemplo. Suponha um ponteiro “p” que armazena um endereço qualquer de memória:

- **p++;** passa a apontar para a próxima posição de memória a partir da posição que p apontava.
- **p+=2;** avança duas posições de memória a partir da posição que p apontava.
- **p--;** retroage uma posição de memória a partir da posição que p aponta.

5. Passagem de parâmetro

Forma de comunicação entre programas e funções, com envio de dados de um para o outro.

- Durante a declaração de uma função, além do nome, devem-se estabelecer os **parâmetros de comunicação**.
- Variáveis declaradas dentro de uma função apresentam **escopo local**.
- Para um programa usar uma função, basta incluir em seu código uma **chamada** para a função desejada.
- A chamada deve **respeitar a ordem e os tipos** dos parâmetros estabelecidos para a comunicação.

6. Passagem de parâmetro por referência

Também conhecida por passagem de parâmetro por endereço.

Tem como vantagens:

- Menor custo de transferência de dados para a função;
- Possibilita que o valor do argumento (variável de quem chamou a função) possa ser alterado pela função.

6. Passagem de parâmetro por referência

Quando um parâmetro é passado por referência, o que é fornecido para a função é o **endereço de memória da variável** correspondente ao parâmetro.

A função chamada recebe o parâmetro (endereço de memória) em um **ponteiro**, encarregado de ser o apontador para o endereço de memória correspondente.

Assim, o parâmetro recebido e o argumento enviado **referenciam o mesmo endereço de memória**.



6. Passagem de parâmetro por referência

Exemplo:

```
void FCalc(int x1, int *x2) {  
    x1 = x1 + 40;  
    *x2 = *x2 - 5;  
}  
  
int main() {  
    int v1, v2;  
    v1= 10;  
    v2= 20;  
    printf("Valores antes da chamada da função\n");  
    printf("Valor1: %d      Valor2: %d", v1, v2);  
  
    FCalc(v1, &v2);  
    printf("Valores após a execução da função\n");  
    printf("Valor1: %d      Valor2: %d", v1, v2);  
}
```

6. Passagem de parâmetro por referência

Exemplo:

```
void FCalc(int x1, int *x2) {  
    x1 = x1 + 40;  
    *x2 = *x2 - 5;  
}
```

Parâmetros :
X1 – por valor
X2 – por referência

void: Não tem retorno de valores

```
int main() {  
    int v1, v2;  
    v1= 10;  
    v2= 20;  
    printf("Valores antes da chamada da função\n");  
    printf("Valor1: %d      Valor2: %d", v1, v2);
```

Argumentos:
v1 – cópia do valor de v1
&v2 – endereço de v2

```
    FCalc(v1, &v2);  
    printf("Valores após a execução da função\n");  
    printf("Valor1: %d      Valor2: %d", v1, v2);  
}
```

7. Exercícios

**Vamos
Programar!**



7. Exercícios

- 1) Usando passagem de parâmetro por referência, faça uma função que receba, por parâmetro, a altura e o sexo de uma pessoa. A função deve alterar, no programa principal, o peso ideal dessa pessoa.
 - ✓ para homens, calcular o peso ideal usando a seguinte fórmula: $\text{peso ideal} = 72.7 \times \text{alt} - 58$; e
 - ✓ para mulheres, $\text{peso ideal} = 62.1 \times \text{alt} - 44.7$.

7. Exercícios

2) Dado dois números inteiros entrados no programa principal, desenvolva uma função que calcule o quociente (inteiro) e o resto da divisão entre esses dois números. Os valores calculados devem ser exibidos no programa principal.

7. Exercícios

- 3) Resolva o problema URI1018 com função e passagem de parâmetros por valor.
- 4) Resolva o mesmo problema com função e passagem de parâmetros por referência.

8. Terceiro momento: síntese

As funções são rotinas que resolvem problemas específicos dentro do programa.

A comunicação de dados entre o código principal e uma função ou entre funções ocorre por meio da passagem de parâmetros.

Na passagem de parâmetros por valor, uma cópia do conteúdo do argumento é encaminhado para o parâmetro da função. Se o conteúdo do parâmetro for alterado, o valor do argumento fica preservado.

Na passagem de parâmetros por referência, o endereço de memória do argumento é encaminhado para o parâmetro da função. Se o conteúdo do parâmetro for alterado, o valor do argumento também é alterado.