

# Engenharia de Computação

## Fundamentos de Programação

### Aula 17 – Funções - Passagem de Parâmetros

**Prof. Muriel de Souza Godoi**  
[muriel@utfpr.edu.br](mailto:muriel@utfpr.edu.br)

# • O Passagem de parâmetros

- **Parâmetros**: Valores a serem processados por uma determinada função
- Na Linguagem C, podem ser de dois tipos
  - Passagem por **cópia** (ou valor);
  - Passagem por **referência** (ou endereço)
- Até agora, os parâmetros vistos na aula anterior eram passados por cópia.
- Vamos revisar e explicar melhor?



# Passagem por cópia

Qual o valor exibido na função main?

```
void incrementa(int valor){
    printf("Valor recebido %d\n", valor);
    valor++;
    printf("Valor incrementado %d\n", valor);
    return;
} //incrementa

int main(){
    int valor = 15;
    incrementa(valor);
    printf("Valor após a função: %d\n", valor);
    return 0;
} //main
```

- A variável original não foi alterada, pois foi feita uma **cópia**
- São posições **diferentes** na memória RAM e possuem escopos diferentes!



Valor após a função: 15

# Passagem por Referência

- E se eu quiser que a função **altere as variáveis** passadas como parâmetro?
  - Passar os **endereços** de memória onde essas variáveis estão armazenadas
  - A função **altera diretamente** o valor
- Lembrando o exemplo anterior
  - **&valor = F060**

Variável	Conteúdo	Endereço
	...	
valor	15	F060
	...	



# Passagem por Referência

- Podemos criar uma variável que armazena um **endereço de memória** que armazena um inteiro;
- Declaramos um ponteiro para **int**

```
int* pointer;
```

- Depois, armazenamos o **endereço** de memória nessa variável ponteiro

```
pointer = &valor;
```

- Podemos, finalmente, acessar o **valor contido** na posição de memória apontado por pointer utilizando:

```
*pointer
```

Variável	Conteúdo	Endereço
	...	
valor	15	F060
pointer	F060	F064
	...	

# Passagem por Referência

- Qual o valor exibido na função main?

```
void incrementa(int* valor){
    printf("Valor recebido %d\n", *valor);
    (*valor)++;
    printf("Valor incrementado %d\n", *valor);
    return;
} //incrementa

int main(){
    int valor = 15;
    incrementa(&valor);
    printf("Valor após a função: %d\n", valor);
    return 0;
} //main
```

- A variável original foi alterada, pois foi passada a **REFERÊNCIA**
- É a **mesma posição** na memória RAM por isso o valor é alterado!



Valor após a função: 16

# Passagem por Referência

- Quando utilizar?
- Na passagem de uma grande quantidade de dados para uma função
  - Economiza **tempo** e **memória**
- Para uma função devolver **mais de um valor**
  - Na Linguagem C, uma função só retorna um valor
  - Passando parâmetros por referência mais valores podem ser gravados como **um meio de saída da função**

# Passagem por Referência - Vetores

- Por padrão, vetores e matrizes são passados por **referência** para as funções
  - **Motivo:** Grande volume de dados copiados. Gasto com processamento e memória
- **Logo:**
  - Uma função recebe apenas o endereço do início do vetor/matriz
  - Qualquer alteração realizada dentro da função é realizada no vetor/matriz original

```
void incrementaVetor(int tam, int vetor[]){  
    for(int i = 0; i < tam; i++){  
        vetor[i]++;  
    }//for  
    return;  
}//incrementaVetor
```



# Passagem de Structs

- **Por padrão**, estruturas são passados por **cópia** para as funções
  - Motivo: As estruturas não costumam ter muitos membros, logo, o gasto extra não é muito grande.
- **Logo:**
  - Uma função recebe uma cópia de toda a estrutura
  - Qualquer alteração realizada dentro da função **não é** realizada no vetor/matriz original
- Mas, e se a estrutura conter vetores e matrizes?
  - A performance pode ser comprometida!
- **Solução?**
  - Passar um **ponteiro para a estrutura** na função!

# Passagem de Structs por Referência

```
typedef struct{
    int x, y;
}Ponto;

void incrementa(Ponto* p){
    p->x++;
    p->y++;
    return;
}//incrementa

int main(){
    Ponto p1 = {1, 2};
    incrementa(&p1);
    printf("( %d, %d)\n", p1.x, p1.y);
    return 0;
}//main
```

- O operador `->` é chamado de **seta** e deve ser utilizada para acessar um membro da estrutura por meio de um ponteiro para a estrutura

- A expressão:

```
p->x++;
```

- É equivalente a:

```
(*p).x++;
```



Utilize **ponto** quando estiver acessando os membros na estrutura  
e **seta** quando tiver um ponteiro para a estrutura

# Exercícios

- 1) Escreva um programa que receba um número inteiro representando a quantidade total de segundos e, usando passagem de parâmetros por referência, converta a quantidade informada de segundos em Horas, Minutos e Segundos. Imprima o resultado da conversão no formato HH:MM:SS. Utilize o seguinte protótipo da função:

```
void converteHora(int total_segundos, int* hora, int* min, int* seg)
```

- 2) Reescreva o exercício anterior utilizando a estrutura horário (contendo hora, minuto e segundo) e passando a estrutura por referência. Utilize o seguinte protótipo da função:

```
void converteHorario(int total_segundos, Horario* hor)
```

# Exercícios

- **3)** Crie um Sistema de Gerenciamento de Bandas seguindo os seguintes passos:
  - **a)** Defina uma estrutura que irá representar bandas de música. Essa estrutura deve ter o nome da banda, que tipo de música ela toca, o número de integrantes e em que posição do ranking essa banda está dentre as suas 5 bandas favoritas;
  - **b)** Crie uma função para preencher as 5 estruturas de bandas criadas no exemplo passado. Após criar e preencher, exiba todas as informações das bandas/estruturas.
  - **c)** Crie uma função que peça ao usuário um número de 1 até 5. Em seguida, seu programa deve exibir informações da banda cuja posição no seu ranking é a que foi solicitada pelo usuário;
  - **d)** Crie uma função que peça ao usuário um tipo de música e exiba as bandas com esse tipo de música no seu ranking.
  - **e)** Crie uma função que peça o nome de uma banda ao usuário e diga se ela está entre suas bandas favoritas ou não;
  - **f)** Agora junte tudo e crie um menu com as opções de preencher as estruturas e todas as opções das questões passadas.