Aula 08:

Funções II

Prof. Edvard

edvard@unifei.edu.br

Universidade Federal de Itajubá

Regras de Escopo

- **Escopo:** “Limite ou abrangência de uma operação” (ex.: ainda não definiram o escopo da campanha). [Dicionário Priberam da Língua Portuguesa]
- **Escopo** é a **parte do programa** onde a variável pode ser utilizada.
  - Quando uma variável é declarada no início do programa, fora das funções e de qualquer bloco de comandos cercado por chaves, seu escopo é global, ou seja, ela pode ser acessada e modificada de qualquer local do programa, em qualquer função.
  - Vejamos um exemplo, onde declaramos uma matriz global e criamos uma função que a inicializa com valores aleatórios de 0 a 9 e outra que a imprime.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Prototipos
void inicializa_matriz();
void imprime_matriz();

// matriz global
int mat[10][10];

int main()
{
    srand(time(NULL));
    inicializa_matriz();
    imprime_matriz();
    return 0;
}

// Gera numeros aleatorios e inicializa
void inicializa_matriz()
{
    int i, j;
    for(i = 0; i < 10; i++)
        for(j = 0; j < 10; j++)
            mat[i][j] = rand() % 10;
}
```

```c
// Formata e imprime matriz
void imprime_matriz()
{
    int i, j;
    for(i = 0; i < 10; i++)
    {
        for(j = 0; j < 10; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

Tanto a função de inicialização quanto a de impressão conseguem acessar a matriz **mat**, pois seu escopo é global.

```
5 3 3 8 3 1 6 9 2 3
5 2 5 1 7 3 1 9 9 2
1 1 5 8 4 1 7 1 4 4
4 5 5 4 2 3 3 0 7 4
2 4 1 4 5 1 8 0 0 2
6 0 2 2 1 9 8 2 0 5
8 4 6 7 9 2 6 6 5 7
8 5 9 1 2 8 2 1 3 1
9 9 4 3 8 3 3 0 1 7
7 3 7 4 3 9 7 7 7 8

Process returned 0 (0x0)
Press any key to continue.
```

Regras de Escopo

- Por outro lado, quando as variáveis são declaradas **dentro de um determinado bloco de comandos**, ela apenas pode ser acessada e modificada localmente, entre a sua declaração e a chave que fecha o bloco (}).
- Seu escopo é limitado ao bloco onde foram declaradas e chamamos de escopo local.
  - Em uma função, normalmente declaramos todas as nossas variáveis locais em primeiro lugar e, a seguir, iniciamos a escrita dos comandos.
  - Imagine que, no exemplo anterior, peçamos ao usuário para que escolha um valor para inicializar toda a matriz.

```c
#include <stdio.h>

// Prototipos
void inicializa_matriz(int);
void imprime_matriz();

// matriz global
int mat[10][10];

int main()
{
    int numero;
    printf("Entre com o valor para inicializar: ");
    scanf("%d", &numero);

    inicializa_matriz(numero); // parametro passado
    imprime_matriz();
    return 0;
}

// Inicializa matriz com valor do parametro
void inicializa_matriz(int n)
{
    int i, j;
    for(i = 0; i < 10; i++)
        for(j = 0; j < 10; j++)
            mat[i][j] = n;
}
```

Nesse caso, a variável número possui um alcance limitado à função onde foi declarada: a main. Escopo é local.

Para que a função de inicialização conheça seu valor, é necessário que a passemos como **parâmetro**.

```c
// Formata e imprime matriz
void imprime_matriz()
{
    int i, j;
    for(i = 0; i < 10; i++)
    {
        for(j = 0; j < 10; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

Parâmetros

- Os parâmetros têm por finalidade servir, portanto, como um **meio de comunicação entre funções.**
  - **Pergunta:** Se todas as variáveis em seu programa fossem globais, haveria necessidade de passagem de parâmetros?
- Em C, os parâmetros de uma função são sempre passados por valor, ou seja, uma cópia do dado é feita e passada para a função.
  - **Lembre-se da aula passada:** as variáveis passadas como parâmetro por valor nunca têm seu conteúdo alterado dentro da função.

```c
#include <stdio.h>

// recebe parâmetro por valor e modifica
void altera_valor(int x)
{
    x = 100;
}

int main()
{
    // inicializa x com o valor 0
    int x = 0;
    printf("Valor de x antes da funcao: %d.\n", x);
    altera_valor(x);

    // O valor de x não foi mudado, e sim de sua cópia
    // cujo escopo está limitado à função altera_valor
    printf("Valor de x antes da funcao: %d.\n", x);
    return 0;
}
```

```
Valor de x antes da funcao: 0.
Valor de x antes da funcao: 0.

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

É uma maneira bastante segura para passagem de parâmetro.

Evita qualquer tipo de **modificação indesejada** que possa ser realizada na variável.

Padrão do C.

Parâmetros

- No entanto, existem casos onde é desejável que a função modifique o valor da variável passada como parâmetro.
  - Passagem por **Referência.**
- Na passagem por referência, não passamos simplesmente o valor da variável para a função: passamos seu endereço na memória.
  - Quando se modifica o valor do parâmetro (a variável local), o valor fica **armazenado no mesmo endereço de memória**.
  - Ao retornar para a função de chamada, o endereço de memória onde foi armazenado o parâmetro conterá o valor modificado.
- Já utilizamos a chamada de uma função com passagem por referência muitas vezes, com uma função bastante conhecida.
  - Alguém arrisca um palpite?

Parâmetros

- Já fizemos chamada com passagem de parâmetro por referência inúmeras vezes com o **scanf**. Veja o exemplo:

```
int x;
printf("Entre com um valor inteiro: ");
scanf("%d", &x); // por isso utilizamos o & comercial
```

- Sempre que é necessário ler alguma coisa do teclado, passamos para a função **scanf** o nome da variável onde o dado será armazenado, precedida de &. Essa variável tem seu valor modificado dentro da função scanf, e seu valor pode ser acessado normalmente na main.
  - O “e comercial” (&) indica exatamente que o que estamos passando não é o valor de x, mas seu **endereço**.
  - Utilizando o **endereço da variável**, qualquer modificação que seja feita na variável é refletida, também, fora da função.

Parâmetros

- Para passar parâmetros por referência, a função precisa utilizar ponteiros.
  - Ponteiros são um tipo especial de variáveis, que armazenam um endereço de memória, da mesma maneira que uma variável comum armazena um valor.
- Para declarar um parâmetro como passagem por referência, devemos incluir um * antes de seu nome. Na chamada da função, devemos passar o endereço da variável (&).
- Exemplo:

```
// valor pode ser modificado
int atualiza_valor(int *valor);
// opcao mode ser mofidicado
void escolhe_opcao(int *opcao);
// n_alunos e exame não podem, mas media pode ser modificada
void update_media(float n_alunos, float exame, float *media);
```

Ao Trabalho!

- Escreva uma função chamada **troca**, que troca os valores de duas variáveis inteiras **a** e **b**. Ao final, a terá o valor inicial de b, e vice-versa.
  - Para completar a tarefa, a passagem dos parâmetros deve ser feita por referência.
- Depois disso, utilize a função no programa principal e imprima os resultados obtidos.

```c
#include <stdio.h>

// prototipo
void troca(int*, int*);

int main()
{
    int a, b;

    printf("Entre com dois valores inteiros: ");
    scanf("%d %d", &a, &b); // chamada por referência (&)

    printf("Antes da troca: a = %d e b = %d \n", a, b);
    troca(&a, &b); // chamada por referência (&)
    printf("Depois da troca: a = %d e b = %d \n", a, b);

    return 0;
}

// Repare que é necessário utilizar os asteriscos (*)
// para indicar que a função recebe os endereços das variáveis
void troca(int *x1, int *x2)
{
    int aux;

    // Toda vez que for necessário acessar o valor do
    // parâmetro dentro da função, teremos que precedê-lo
    // de um asterisco (*).
    // Lembre-se de que x1 e x2 são endereços de memória.
    aux = *x1;
    *x1 = *x2;
    *x2 = aux;
}
```

A chamada por referência é sempre feita com "**E comercial**" &.

A declaração dos parâmetros utiliza o **asterisco** (*) para indicar a passagem por referência.

Para acessar valor ao invés de endereço, de-referenciamos o parâmetro, utilizando também o **asterisco** (*).

Ao Trabalho!

- Escreva uma função que receba dois números inteiros a e b, coloque-os em ordem, de maneira que a contenha sempre o menor valor e, ao final, retorne o maior valor entre os dois.
  - Utilize passagem de parâmetros por referência para completar a tarefa.
  - Teste a função no programa principal.

```c
#include <stdio.h>

// prototipo
int ordena(int*, int*);

int main()
{
    int a, b, maior;

    printf("Entre com dois valores inteiros: ");
    scanf("%d %d", &a, &b); // chamada por referência (&)

    printf("Antes da ordenacao: a = %d e b = %d \n", a, b);
    maior = ordena(&a, &b); // chamada por referência (&)
    printf("Depois da ordenacao: a = %d e b = %d \n", a, b);
    printf("O maior valor entre eles eh %d. \n", maior);

    return 0;
}

// coloca os parametros em ordem
int ordena(int *x1, int *x2)
{
    int aux;

    if(*x1 > *x2)
    {
        aux = *x2;
        *x2 = *x1;
        *x1 = aux;
    }

    return *x2;
}
```

```
Entre com dois valores inteiros: 20 10
Antes da ordenacao: a = 20 e b = 10
Depois da ordenacao: a = 10 e b = 20
O maior valor entre eles eh 20.
```

Lembre-se de que a função apenas pode retornar um único valor.

A **passagem por referência** torna possível obter mais valores como retorno da função, modificando o próprio parâmetro.

Ao Trabalho!

- Imagine que temos um objeto movendo-se com aceleração constante em linha reta. Temos o tempo gasto por ele, sua velocidade inicial e o valor da aceleração.
  - Desenvolva uma função em C que retorne através dos parâmetros passados por referência os valores da **velocidade final** e a **distância percorrida** durante o percurso.

$$\Delta s = v_0 t + \frac{a t^2}{2} \qquad v^2 = {v_0}^2 + 2a\Delta s$$

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

// Prototipo
void mruv(float, int, float, float*, float*);

int main()
{
    float acel, vini, vfin, dist;
    int tempo;

    printf("Entre com a aceleracao (m/s2) e a velocidade inicial (m/s): ");
    scanf("%f %f", &acel, &vini);

    printf("Entre com o tempo gasto (s): ");
    scanf("%d", &tempo);

    mruv(acel, tempo, vini, &vfin, &dist);

    printf("Velocidade Final: %.2f m/s\n", vfin);
    printf("Distancia Percorrida: %.2f m\n", dist);

    return 0;
}

void mruv(float a, int t, float vi, float *vf, float *s)
{
    // calculando distancia
    *s = vi*t + a*pow(t, 2)/2;
    *vf = sqrt(vi*vi + 2*a*(*s));
}
```

Os dois valores calculados podem ser aproveitados no programa principal, mesmo não sendo retornados através do comando **return**.

```
Entre com a aceleracao (m/s2) e a velocidade inicial (m/s): 10 0
Entre com o tempo gasto (s): 30
Velocidade Final: 300.00 m/s
Distancia Percorrida: 4500.00 m
```

Parâmetros

- **Resumo:**
  - Passagem de parâmetros por VALOR
    - Parâmetros declarados sem asterisco (*) e recebem cópias dos valores das variáveis passadas (sem &).
    - O processamento realizado por uma função em um parâmetro passado por valor nunca é refletido na variável especificada na chamada.
  - Passagem de parâmetros por REFERÊNCIA
    - Parâmetros declarados com asterisco (*) e recebem o endereço das variáveis especificadas na chamada (com &).
    - O processamento realizado em um parâmetro passado por referência reflete na variável especificada na chamada.
    - Para acessar o seu valor, é necessário colocar um asterisco (*) antes de seu nome (de-referência).