

INTRODUÇÃO



LINGUAGEM C
(PARTE 3)

DIA 4

TÓPICOS DE HOJE



- RECURSÃO
- PONTEIROS
- FUNÇÃO COM PASSAGEM DE PARÂMETROS
 - POR VALOR
 - POR REFERÊNCIA

RECURSÃO

Conceito

- Recursão é uma técnica que permite uma função chamar a si mesma, com o intuito de dividir uma tarefa em subtarefas, facilitando-se a execução do algoritmo e evitando uso de laços e repetições.
- A recursão baseia-se no princípio da indução finita (PIF), no qual há um caso base para um valor inicial e uma equação ou algoritmo para obter o resultado de outros valores, a partir do caso base.

RECURSÃO

Exemplo

- Por exemplo, a função a seguir tem o objetivo de somar todos os números inteiros de um intervalo de 0 a um número inteiro escolhido e mostrar o resultado:

RECURSÃO

```
1 int soma(int k);
2
3 < void main() {
4     int resultado = soma(10);
5     printf("%d", resultado);
6     // Resultado = 55
7 }
8
9 < int soma(int k) {
10 <   if (k > 0) {
11       return k + soma(k - 1);
12 <   } else {
13       return 0;
14 <   }
15 }
```

RECURSÃO

Exemplo

- No exemplo mostrado, o caso recursivo é $\text{soma}(k) = k + \text{soma}(k-1)$ e o caso base é quando $k=0$, sendo $\text{soma}(0) = 0$. Sendo $k = 10$, o computador processará a função desta forma:

soma(10)

10 + soma(9)

10 + (9 + soma(8))

10 + (9 + (8 + soma(7)))

3

10 + (9 + (8 + (7 + (6 + (5 + (4 + (3 + (2 + (1 + (soma(0)))))))))))

$$10 + (9 + (8 + (7 + (6 + (5 + (4 + (3 + (2 + (1 + (0)))))))))) = 55$$

RECURSÃO

Exercício

- Escreva uma função recursiva que determine quantas vezes um dígito K ocorre em um número natural N. Por exemplo, o dígito 3 ocorre 4 vezes em 356337383
 - Dicas:
 - Verifique recursivamente algarismo por algarismo de N e pare quando N não tiver mais algarismos para analisar
 - Use o tipo long para a variável N em vez de int

PONTEIROS

Endereçamento de memória

- Para entender ponteiros em C é fundamental conhecer sobre o conceito de endereçamento de memória.
- Resumidamente, cada operando(variável) e instrução em uma algoritmo é guardado na memória do computador em uma célula(espaço/caixinha) específica e cada célula possui um endereço para que o computador possa encontrá-la e utilizar a informação que está sendo armazenada.

PONTEIROS

Endereçamento de memória

- Exemplo ilustrativo:

Endereço de memória	Célula de memória
*100	a = 5.
*101	b = 8.
*102	a + b.

PONTEIROS

Conceito

- O ponteiro em C tem a ação de apontar para um endereço de memória, que pode estar armazenando um valor e uma variável.
- Exemplo ilustrativo, sendo '*c' o ponteiro:

Endereço de memória	Célula de memória
*100	a = 5.
*101	b = 8.
*102	a + b.
*103	*c -> *100 (a).

PONTEIROS

Aplicação

- Para se declarar um ponteiro é necessário declarar o tipo de dado que ele aponta e colocar um asterisco(*) antes de seu nome.
- Já para referenciar o endereço de memória de uma variável usa-se o símbolo '&' antes da variável.
- No exemplo do próximo slide mostra-se a declaração e aplicação de um ponteiro. Note que o endereço apontado pelo ponteiro e da variável apontada são iguais.

PONTEIROS

```
3 int minhaIdade = 19; // Variavel
4
5 // Declaracao do ponteiro A que aponta para o endereço de memoria da variavel minhaIdade
6 int * A = & minhaIdade ;
7
8 // Resultado da variavel minhaIdade (19)
9 printf ( " % d \n " , minhaIdade ) ;
10
11 // Resultado do endereço de memoria de minhaIdade (0x7ffe5367e044)
12 printf ( " % p \n " , & minhaIdade ) ;
13
14 // Resultado do endereço de memoria de minhaIdade através do ponteiro A (0x7ffe5367e044)
15 printf ( " % p \n " , A ) ;
16
17 // Valor da variavel minhaIdade através do ponteiro A (19)
18 printf ( " % d \n " , * A ) ;
```

PONTEIROS

```
4 int minhaIdade = 19;
5 int idadeVizinho = 35;
6 int * A = & minhaIdade ; // O ponteiro aponta para minhaIdade
7
8 // Resultado do endereco de memoria de minhaIdade atraves do ponteiro A(0 x7ffe5367e044 )
9 printf ( " % p \n " , A ) ;
10
11 // Valor da variavel minhaIdade atraves do ponteiro A (19)
12 printf ( " % d \n " , * A ) ;
13
14 // Valor de minhaIdade alterado
15 minhaIdade = 30;
16
17 // Resultado do endereco de memoria de minhaIdade atraves do ponteiro A (0x7ffe5367e044 )
18 printf ( " % p \n " , A ) ;
19
20 // Valor da variavel minhaIdade atraves do ponteiro A (30)
21 printf ( " % d \n " , * A ) ;
```

PONTEIROS

- Continuando-se o exemplo anterior, neste caso muda-se a variável que o ponteiro aponta, perceba que o endereço referenciado também muda

```
25 A = &idadeVizinho; // O ponteiro agora aponta para idadeVizinho
26
27 // Resultado do endereco de memoria de idadeVizinho atraves do ponteiro A (0x7ffe5367e198)
28 printf ( " % p \n " , A ) ;
29
30 // Valor da variavel idadeVizinho atraves do ponteiro A (35)
31 printf ( " % d \n " , * A ) ;
```

PONTEIROS

Aplicação

- Através de ponteiro é possível realizar operações aritméticas também

```
int x = 4;  
int *p = &x;  
*p *= 2;  
printf("\n %d", x); // x = 8  
printf("\n %d", *p); // *p = 8
```

```
int y = 3 * *p;  
printf("\n %d", y); // y = 24
```

PONTEIROS

Aplicação

- Os ponteiros não apenas referenciam números eles podem referenciar strings também

```
char a[] = {"World"};
char* b = a;
char c[] = {"Hello "};

printf("\n%s %s", c, b); // Hello World
```

PONTEIROS

Aplicação

- Outras propriedades dos ponteiro é que mais de um ponteiro pode apontar para a mesma variável, e que um ponteiro pode apontar para o outro.
- Exemplo:

```
33 int peso = 19;  
34 int * B = & peso ;  
35 int * C = & peso ;  
36 int ** D = & B ;|
```

PONTEIROS

Aplicação

- Quando um ponteiro aponta para um array, o endereço de memória que o ponteiro referencia é equivalente ao endereço de memória do primeiro elemento do array.
- Para apontar aos outros elementos do array soma-se o índice equivalente do elemento ao ponteiro.
- Exemplo no slide a seguir:

PONTEIROS

```
3 int numeros[4] = {25, 50, 75, 100};  
4 int *ponteiro = &numeros; //Ponteiro apontando para o array.  
5  
6 // Endereço de memória apontado pelo ponteiro (0x0061FF0C)  
7 printf("%p\n", ponteiro);  
8  
9 // Valor armazenado no endereço de memória apontado (25)  
10 // Perceba que 25 é o primeiro elemento do array numeros  
11 printf("%d\n", *ponteiro);  
12  
13 // Faz o ponteiro apontar para a célula de memória seguinte,  
14 // que neste caso é o segundo elemento do array  
15 ponteiro += 1;  
16  
17 // Endereço de memória do apontado pelo ponteiro (0x0061FF10)  
18 printf("%p\n", ponteiro);  
19 // Valor armazenado no endereço de memória apontado (50)  
20 printf("%d\n", *ponteiro);
```

PONTEIROS

```
char x[] = {"Hello World"};
char* y = x;
// y = 'H'
printf("\n%s", y); // Hello World
y += 1;
// y = 'e'
printf("\n%s", y); // ello World
```

PONTEIROS

```
3 int numeros[4] = {25, 50, 75, 100};  
4 int *ponteiro = &numeros; //Ponteiro apontando para o array.  
5  
6 // Endereço de memória apontado pelo ponteiro (0x0061FF0C)  
7 printf("%p\n", ponteiro);  
8  
9 // Valor armazenado no endereço de memória apontado (25)  
10 // Perceba que 25 é o primeiro elemento do array numeros  
11 printf("%d\n", *ponteiro);  
12  
13 // Faz o ponteiro apontar para a célula de memória seguinte,  
14 // que neste caso é o segundo elemento do array  
15 ponteiro += 1;  
16  
17 // Endereço de memória do apontado pelo ponteiro (0x0061FF10)  
18 printf("%p\n", ponteiro);  
19 // Valor armazenado no endereço de memória apontado (50)  
20 printf("%d\n", *ponteiro);
```

PONTEIROS

Exercício

- Faça uma função que receba como parâmetro uma string e através de um ponteiro dentro da função, contabilize quantas vogais tem na string.
- Não esqueça que a linguagem trata maiúsculas e minúsculas como caracteres diferentes.
- Dica: Para mover o ponteiro pela string e parar quando chegar no fim da string, utilize um `while(*ponteiro)` e vá incrementando o ponteiro.

PASSAGEM DE PARÂMETROS

Por Valor

- Funções com passagem de parâmetros por valor é o tipo de função que já foi vista e aplicada nos exercícios e slides anteriores
- Trata-se de uma função a qual seu parâmetro recebe uma variável declarada fora dela e produz uma cópia desta variável dentro dela. Se a função alterar o valor da variável, será modificada apenas a cópia da variável dentro dela e não a variável declarada fora.
- Veja o exemplo a seguir:

PASSAGEM DE PARÂMETROS

```
3 // Função que multiplica um número por dois
4 void multiplicadordois(int valor){
5     valor = valor * 2;
6     printf("\nResultado dentro da função multiplicadordois(): %d", valor);
7     printf("\nEndereço de memória dentro da função: %p", &valor);
8 }
9 void main()
10 {
11     int a = 10; // variável
12     multiplicadordois(a); // função recebe a variável 'a' como parâmetro
13     // Resultado dentro da função multiplicadordois(): 20
14     // Endereço de memória dentro da função: 0x0061FE00
15
16
17     printf("\nValor fora da função multiplicadordois(): %d", a); // a = 10
18     printf("\nEndereço de memória fora da função: %p", &a); // a = 0x0061FF1C
19
20 }
```

PASSAGEM DE PARÂMETROS

Por Valor

- Como pode se notar no exemplo anterio a variável 'a' é alterada dentro da função multiplicadordois(), mas não fora da função.
- Percebe-se também que o símbolo 'a' dentro e fora da função tem endereços de memória diferentes, ou seja, o computador as trata como duas variáveis diferentes.

PASSAGEM DE PARÂMETROS

Por referência

- Na função por passagem por referência, a função recebe o endereço de memória de uma variável como parâmetro.
- Neste caso, se a variável declarada fora da função tiver seu valor alterado dentro dela, a variável fora da função também tem seu valor alterado.
- Utiliza-se ponteiros para acessar o valor que está no endereço de memória
- Veja o exemplo a seguir:

PASSAGEM DE PARÂMETROS

```
4 // Função que multiplica um numero por 2.
5 void multiplicadordois(int *valor){
6     *valor = *valor * 2;
7     printf("\n Resultado da variavel dentro da função multiplicadordois(): %d", *valor);
8     printf("\nEndereco de memoria dentro da função: %p", valor);
9 }
10
11
12 void main(){
13     int a = 10;
14
15     multiplicadordois(&a); // função recebe o endereço de memória de 'a' como parâmetro.
16     // Resultado da variavel dentro da função multiplicadordois(): 20
17     // Endereco de memoria dentro da função: 0x0061FF1C
18
19     printf("\n Resultado da variavel fora da função multiplicadordois(): %d", a); //20
20     printf("\nEndereco de memoria fora da função: %p", &a); //0x0061FF1C
21 }
```

PASSAGEM DE PARÂMETROS

Por referência

- Como pode se notar no exemplo anterior a variável ‘a’ é alterada dentro da função multiplicadordois() e fora da função.
- Percebe-se também que o símbolo ‘a’ dentro e fora da função tem os mesmos endereços de memória, ou seja, o computador trata ‘a’ como a mesma variável, dentro e fora da função.
- Isto ocorreu pois a função recebeu o endereço de memória da variável e modificou seu valor através de um ponteiro que a referencia. Basicamente acessando a variável diretamente em vez de fazer uma cópia.

PASSAGEM DE PARÂMETROS

Exercício

- Faça uma função que ordene um vetor de números inteiros desordenado
- Tente fazer por passagem por parâmetro e por referência
- Não pode usar função pronta, tente usar repetições e condicionais para a ordenção
- **Dicas:** para a função auxiliar passe como parâmetro o vetor e o tamanho do vetor
- Para o tamanho do vetor use: `int tamanho = sizeof(vetor) / sizeof(vetor[0])`

ULTIMOS EXERCÍCIO

Exercício

- Faça uma função que mostre todos os números primos até um número escolhido. Números primos são aqueles números que seus divisores são apenas 1 e ele mesmo. Ex: 2, 3, 5, 7, 11, 13, 17, 19, 23...
- Se quiser resolver o exercício utilizando raiz quadrada, use a biblioteca `#include <math.h>` e use a função `double y = sqrt(x)` para encontrar a raiz.

ULTIMOS EXERCÍCIO

Exercício

- Faça uma função que receba dois vetores de números inteiros e devolva um vetor com todos os números presentes nos dois vetores sem repetição, ou seja, não pode haver dois números iguais no novo vetor.
- Dica: para o novo vetor use como o tamanho a soma do tamanho dos dois vetores iniciais. Não importa se o novo vetor tenha alguns espaços vazios.

ULTIMOS EXERCÍCIO

Exercício

- Faça uma ou mais funções para verificar se uma string é um palíndromo,, ou seja se seu texto normal e invertido são iguais
 - Exemplo: texto: "Hello World" , invertido:"dlrow olleH" . Não são palíndromos
 - Texto: "reviver", invertido: "reviver". São palíndromos.
-
- Dicas: Existe diferença entre maiúsculas e minúsculas no C.
 - Calculo do tamanho de string: `sizeof(string) / sizeof(string[0]) - 1;`

ULTIMOS EXERCÍCIO

Exercício

- Faça uma função recursiva que multiplique todos os elementos ímpares de um vetor entre si e retorne o resultado final da multiplicação.
- Faça por passagem de parâmetro por valor e depois por referência.
- Se conseguiu resolver dos dois modos tente modificar a função de passagem por valor, usando um ponteiro para referenciar o vetor e percorre-lo recursivamente. Utilize a referência a vetor :
`int* ponteiro = (int*)(vetor); Buscar próximo elemento: (int*)(ponteiro +1)`

EXTRA

Sintaxes adicionais

- Neste workshop foi passado a sintaxe básica da linguagem C, porém algumas outras propriedades da linguagem não foram mostradas por falta de tempo e por serem mais complexas, principalmente para quem está iniciando na linguagem.
- O slide a seguir mostrará alguns tópicos da linguagem que não foram vista aqui, mas que você pode estudar e se aprofundar depois.

EXTRA

Sintaxes adicionais

- Estrutura (struct)
- Enumeração(enum)
- Alocação de memória(calloc, malloc, realloc, free)
- Arquivos(fopen, fclose)
- Manipulação de bits(&, |, ^, <<, >>)

EXTRA

Sintaxes adicionais

- Links para de materiais para estudar as outras propriedades
- <https://www.w3schools.com/c/index.php>
- <https://embarcados.com.br/bits-em-linguagem-c/>

CONGRATULATIONS



MUITO OBRIGADO



DIN Departamento de
Informática

