

Aula 4

Linguagem de Programação

Prof. Sandro de Araujo, MSc.

▪ Ponteiros – STRUCT, FUNÇÃO, ALOCAÇÃO DINÂMICA

Conversa Inicial

▪ Essa aula apresenta a seguinte estrutura de conteúdo:

- PONTEIRO – STRUCT
- STRUCT DE PONTEIROS
- STRUCT – FUNÇÃO
- ALOCAÇÃO DINÂMICA – CALLOC() E FREE()
- ALOCAÇÃO DINÂMICA – MALLOC() E REALLOC()

- O objetivo desta aula é conhecer os principais conceitos e aplicações de ponteiros em struct, struct de ponteiros, struct com funções, alocação dinâmica de memória com as funções calloc(), free(), malloc() e realloc(), na linguagem C para resolver problemas computacionais

PONTEIRO – STRUCT

PONTEIRO – STRUCT

- Na linguagem C, uma struct é uma coleção de variáveis referenciada pelo mesmo nome, conhecida também como tipo de dado agregado
- Uma vez que variáveis do tipo estrutura são tratadas exatamente da mesma forma que variáveis de tipos básicos, é possível definir variáveis do tipo ponteiro para estruturas

PONTEIRO – STRUCT

Sintaxe

```
struct<nome_da_struct>*<nome_do_ponteiro>;
```

PONTEIRO – STRUCT

Operador & (e comercial) "endereço de"

- O seu resultado sempre será o endereço de memória do objeto em questão
- Normalmente referencia o local onde uma variável está alocada na memória. Isto é, o operador & cria um ponteiro

Operador * (Asterisco) "o valor apontado por"

- Indica o componente da struct que pode ser modificado diretamente na memória
- Esse operador faz o compilador entender que aquela variável vai guardar um endereço para aquele tipo especificado

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

```
int main(){
```

```
    struct calendario{
        int dia;
        int mes;
        int ano;
    }; struct calendario agora, *depois;
```

```
depois = &agora;
```

```
/* (*depois).dia = 28;
   (*depois).mes = 09;
   (*depois).ano = 2018; */
```

```
depois->dia = 28;
depois->mes = 9;
depois->ano = 2018;
```

```
28/9/2018
Pressione qualquer tecla para continuar. . .
```

Struct de ponteiros

Strucut de Ponteiros

- Ponteiros também podem ser definidos como componentes de estruturas, basta colocar o operador '*' asterisco antes dos componentes de uma struct

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

int main(){

    struct calendario{
        int *dia;
        int *mes;
        int *ano;
    }; struct calendario atual;
```

```
int diaSetembro = 28;
int mesSetembro = 9;
int anoSetembro = 2018;

atual.dia = &diaSetembro;
atual.mes = &mesSetembro;
atual.ano = &anoSetembro;
```

```
Endereco diaSetembro = 000000000062FE2C
*dia aponta para =    000000000062FE2C

Endereco mesSetembro = 000000000062FE28
*mes aponta para =    000000000062FE28

Endereco anoSetembro = 000000000062FE24
*Ano aponta para =    000000000062FE24

Dia: 28 Mes: 9 Ano: 28

Pressione qualquer tecla para continuar. . . _
```

Struct – Função

Struct - Função

- Assim como uma variável, uma struct também pode ser passada como parâmetro para uma função e essa passagem é feita de duas formas:

- Por valor
- Por referência

Passagem de uma struct por valor

- Uma cópia do componente da struct é usada e alterada dentro da função sem afetar a variável da estrutura, na memória, da qual ela foi gerada

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

```
struct p_valor{
    int a,b;
};
```

```
void imprimir_soma_valor(int num);
```

```
int main(){
    //Passagem de um campo por valor
```

```
    struct p_valor pont1 = {201, 302};
```

```
printf("VALORES ANTES DA CHAMADA DA FUNCAO\n");
printf("Valor do componente 'a' = %d \n", pont1.a);
printf("Valor do componente 'b' = %d \n\n", pont1.b);

printf("VALORES NA DA CHAMADA DA FUNCAO\n");
imprimir_soma_valor(pont1.a); //Faz uma cópia de a
imprimir_soma_valor(pont1.b); //Faz uma cópia de b

printf("\nVALORES DEPOIS DA CHAMADA DA FUNCAO\n");
printf("Valor do componente 'a' = %d \n", pont1.a);
printf("Valor do componente 'b' = %d \n\n", pont1.b);

system("pause");
return 0;
}
```

①

```
VALORES ANTES DA CHAMADA DA FUNCAO
Valor do componente 'a' = 201
Valor do componente 'b' = 302

VALORES NA DA CHAMADA DA FUNCAO
Valor 201 alterado na funcao para= 204
Valor 302 alterado na funcao para= 305

VALORES DEPOIS DA CHAMADA DA FUNCAO
Valor do componente 'a' = 201
Valor do componente 'b' = 302

Pressione qualquer tecla para continuar. . . _
```

Passagem de uma struct por referência

- Alterações dos parâmetros sofridas dentro da função também serão sentidas fora dela, ou seja, os dados são alterado diretamente na memória

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

void imprimir_soma_valor(int *num);

int main(){
    struct p_valor{
        int a,b;
    }; struct p_valor x, *px;

    px = &x;

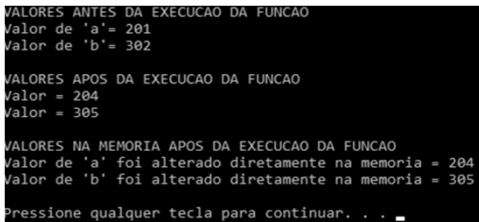
    px->a=201;
    px->b=302;
```

```
printf("VALORES ANTES DA EXECUCAO DA FUNCAO\n");
printf("Valor de 'a'= %d \n", px->a);
printf("Valor de 'b'= %d \n", px->b);

printf("VALORES APOS DA EXECUCAO DA FUNCAO\n");
imprimir_soma_valor(&x.a); //Passa o endereco de 'a' para a função
imprimir_soma_valor(&x.b); //Passa o endereco de 'b' para a função

printf("\nVALORES NA MEMORIA APOS DA EXECUCAO DA FUNCAO");
printf("\nValor de 'a' foi alterado diretamente na memoria = %d \n",
x.a);
printf("Valor de 'b' foi alterado diretamente na memoria = %d \n",
x.b);

//printf("\nValor de 'a' foi alterado diretamente na memoria = %d \n",
px->a);
//printf("Valor de 'a' foi alterado diretamente na memoria = %d \n",
px->b);
```



A terminal window with a dark background and light text. It shows the output of the program. The first section, 'VALORES ANTES DA EXECUCAO DA FUNCAO', shows 'a' as 201 and 'b' as 302. The second section, 'VALORES APOS DA EXECUCAO DA FUNCAO', shows 'a' as 204 and 'b' as 305. The third section, 'VALORES NA MEMORIA APOS DA EXECUCAO DA FUNCAO', shows 'a' as 204 and 'b' as 305. The prompt 'Pressione qualquer tecla para continuar...' is at the bottom.

```
VALORES ANTES DA EXECUCAO DA FUNCAO
Valor de 'a'= 201
Valor de 'b'= 302

VALORES APOS DA EXECUCAO DA FUNCAO
Valor = 204
Valor = 305

VALORES NA MEMORIA APOS DA EXECUCAO DA FUNCAO
Valor de 'a' foi alterado diretamente na memoria = 204
Valor de 'b' foi alterado diretamente na memoria = 305

Pressione qualquer tecla para continuar. . . _
```

Alocação Dinâmica CALLOC() e FREE()

- A alocação dinâmica de memória é um mecanismo que reserva uma quantidade de memória, em região conhecida como *heap*, durante a execução de um programa
- Na biblioteca *stdlib* temos quatro funções:
 - calloc
 - free
 - malloc
 - realloc

- A função `calloc()`
- Cria um vetor com tamanho dinâmico
- Serve para alocar memória durante a execução do programa
- Faz o pedido de memória ao computador e retorna um ponteiro com o endereço do início do espaço alocado
- Preenche com zero todos os bits

`void calloc(unsigned int nElementos, unsigned int tElemento);`*

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

int main()
{
    int *px;

    px = (int*) calloc(4, sizeof(int));
    if(px == NULL){
        printf("ERRO! Não tem memória suficiente.");
        exit(1); // finaliza o programa.
    }
}
```

```
① Digite px[0]: 5
    Digite px[1]: 6
    Digite px[2]: 8
    Digite px[3]: 9

    Posicao px[0]= 5
    Posicao px[1]= 6
    Posicao px[2]= 8
    Posicao px[3]= 9

    Pressione qualquer tecla para continuar. . .
```

A função `free()`

- Libera o espaço de memória que foi previamente alocado
- Para declarar uma função `free()` usa-se a sintaxe mostrada abaixo
- `void free(void *nomePonteiro);`

```
int main()
{
    px = (int*) calloc(4, sizeof(int));
    if(px == NULL){
        printf("ERRO! Não tem memória suficiente.");
        exit(1); // finaliza o programa.
    }

    ...

    free(px);
    system("pause");
    return 0;
}
```

Alocação Dinâmica `MALLOC()` e `REALLOC ()`

A função malloc()

- ▀ Parecida com a função calloc();
- ▀ Com uma grande diferença, essa função não coloca zero nos bits do espaço alocado
- ▀ Para declarar uma função malloc() usa-se a sintaxe mostrada abaixo
- ▀ `void* malloc(unsigned int nElementos);`

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

int main()
{
    int *px;

    px = (int*) malloc(4 * sizeof(int));
    if(px == NULL){
        printf("ERRO! Não tem memória suficiente.");
        exit(1); // finaliza o programa.
    }
}
```

```
int i;

for(i=0; i<4; i++){
    printf("Digite px[%d]: ", i);
    scanf("%d", &px[i]);
}

printf("\n");

for(i=0; i<4; i++){
    printf("Posicao px[%d]= %d\n", i
,px[i]);
}
```

```
① Digite px[0]: 45
    Digite px[1]: 52
    Digite px[2]: 78
    Digite px[3]: 44

Posicao px[0]= 45
Posicao px[1]= 52
Posicao px[2]= 78
Posicao px[3]= 44

Pressione qualquer tecla para continuar. . .
```

A função realloc()

- ▀ Aloca e realoca um espaço na memória durante a execução do programa
 - ▀ Essa função realiza um pedido de memória e retorna um ponteiro com o endereço do início do espaço de memória alocado
 - ▀ Para declarar uma função realloc() usa-se a sintaxe mostrada abaixo
- `void* realloc(void* nomePonteiro, unsigned int nElementos);`

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

int main(){

    int *px = (int*) malloc(25 * sizeof(int));
    px = (int*) realloc(px, 50 * sizeof(int));

    system("pause");
    return 0;
}
```