Anhanguera

Aqui o seu esforço ganha força.

Aula 05 – Introdução à Alocação Dinâmica de Memória

Prof. Esp. Rodrigo Hentz

Alocação Dinâmica

- Às vezes, a quantidade de memória a alocar só se torna conhecida durante a execução do programa. Ex: Quantos alunos preciso alocar? Quantos itens serão ineridos? Quantos cadastros?
- Para lidar com essa situação é preciso recorrer à alocação dinâmica de memória.
- A alocação dinâmica é gerenciada por um conjunto de funções que estão na biblioteca ***stdlib***.
- Para usar esta biblioteca, é preciso colocar no início do programa:

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

Alocação Dinâmica

- A alocação dinâmica é o processo que aloca memória em tempo de execução.

- Ela é utilizada quando não se sabe ao certo quanto de memória será necessário para o armazenamento das informações, podendo ser determinadas em tempo de execução conforme a necessidade do programa.

- Dessa forma evita-se o desperdício de memória.

Alocação Dinâmica

- Ao utilizar alocações dinâmicas é preciso SEMPRE liberar a memória após o término de seu uso.

- Quando se utiliza alocações estáticas, que vimos até agora, quem faz este processo é o próprio compilador.
- Na alocação dinâmica, deve ser feito pelo programador.

Alocação Dinâmica

- As principais funções utilizadas para trabalhar com alocações dinâmicas são:

- sizeof()
- malloc()
- calloc()
- free()

SIZEOF

- Retorna o tamanho que um tipo de variável ocupa na memória, inclusive estruturas.
- Sintaxe:

```
sizeof(<tipo_dado>)
```

Anhanguera

SIZEOF

```
typedef struct
{
    char nome[30];
    int idade;
    float salario;
} sFuncionario;

int main(int argc, char *argv[]) {

    printf("Tamanho de uma variavel inteira: %d bytes.\n", sizeof(int));
    printf("Tamanho de uma variavel char:    %d bytes.\n", sizeof(char));
    printf("Tamanho de uma variavel float:   %d bytes.\n", sizeof(float));
    printf("Tamanho de uma variavel double:  %d bytes.\n", sizeof(double));
    printf("Tamanho de uma variavel sFuncionario:  %d bytes.\n", sizeof(sFuncionario));

    return 0;
}
```

MALLOC

- A função malloc aloca um espaço de memória e retorna um ponteiro do local alocado.
- Sintaxe:

```
(<tipo>*)malloc(<tamanho>)
```

Anhanguera

MALLOC

- Podemos usar da seguinte maneira:
- Passar um tamanho á definido para a função:

```
(char*)malloc(1)
```

- Passar o retorno da função sizeof como parâmetro:

```
(int*)malloc(sizeof(int))
```

Anhanguera

MALLOC

- Exemplo

```
char* pchar;
pchar = (char*)malloc(1);
printf("\nInsira o caracter: ");
scanf( "%c", pchar);
printf("\nValor do caracter: %c", *pchar);
```

- As duas últimas linhas estão corretas?
- Função ***scanf*** e ***printf*** não precisam ter o ***&*** no parâmetro?

MALLOC

- Exemplo no caso de uma estrutura

```
typedef struct
{
    char nome[30];
    int idade;
    float salario;
} sFuncionario;

sFuncionario* pFuncionario =
    (sFuncionario*)malloc(sizeof(sFuncionario));
pFuncionario->idade = 10;
strcpy(pFuncionario->nome, "Maria");
pFuncionario->salario = 3000;
printf("\nFuncionario Idade: %s", pFuncionario->nome);
printf("\nFuncionario Idade: %d", pFuncionario->idade);
printf("\nFuncionario Salario: %.2f", pFuncionario->salario);
```

MALLOC

- Exemplo no caso de um vetor

```
int* pvetor = (int*)malloc(10 * sizeof(int));
pvetor[0] = 10;
pvetor[9] = 90;
printf("\nVetor posicao 0: %d", pvetor[0]);
printf("\nVetor posicao 9: %d", pvetor[9]);
return 0;
```

Verificação do espaço alocado

- Verificando se o espaço foi alocado antes de atribuir valores.
- Sintaxe:

```
if (ponteiro != NULL)
{
        /* instruções */
}
else /* mensagem de erro */
```

Verificação do espaço alocado

- Exemplo:

```
int* pvetor2 = (int*)malloc(10 * sizeof(int));
if (pvetor2 != NULL)
{
    printf("\nVetor alocado");
    pvetor2[3] = 30;
    pvetor2[6] = 60;
    printf("\nVetor posicao 3: %d", pvetor2[3]);
    printf("\nVetor posicao 6: %d", pvetor2[6]);
} else printf("\nVetor nao alocado");
```

FREE

- Libera o espaço de memória alocado dinamicamente.
- Sintaxe:

```
free(<ponteiro>)
```

FREE

- Exemplo:

```
int* pvetor2 = (int*)malloc(10 * sizeof(int));
if (pvetor2 != NULL)
{
    printf("\nVetor alocado");
    pvetor2[3] = 30;
    pvetor2[6] = 60;
    printf("\nVetor posicao 3: %d", pvetor2[3]);
    printf("\nVetor posicao 6: %d", pvetor2[6]);
} else printf("\nVetor nao alocado");

free(pvetor2);
```

REALLOC

- Aumenta a memória alocada por um determinado tipo. Por exemplo, temos 10 posições alocadas e durante a execução do programa é necessário criar mais duas posições.

- Sintaxe:

```
realloc(<ponteiro>, <tamanho>)
```

REALLOC

- Exemplo.

```
int* pvetor3 = (int*)malloc(1 * sizeof(int));
if (pvetor3 != NULL)
{
    printf("\nVetor alocado");
    pvetor3[0] = 10;
    printf("\nVetor posicao 0: %d", pvetor3[0]);
    pvetor3 = (int*)realloc(pvetor3, 4 * sizeof(int));
    if(pvetor3 != NULL)
    {
        pvetor3[1] = 11; pvetor3[2] = 12; pvetor3[3] = 13;
        printf("\nVetor posicao 0: %d", pvetor3[0]);
        printf("\nVetor posicao 1: %d", pvetor3[1]);
        printf("\nVetor posicao 2: %d", pvetor3[2]);
        printf("\nVetor posicao 3: %d", pvetor3[3]);
    }
    else printf("\nVetor nao realocado");
} else printf("\nVetor nao alocado");
free(pvetor3);
```

No código abaixo quais seriam os valores impressos?

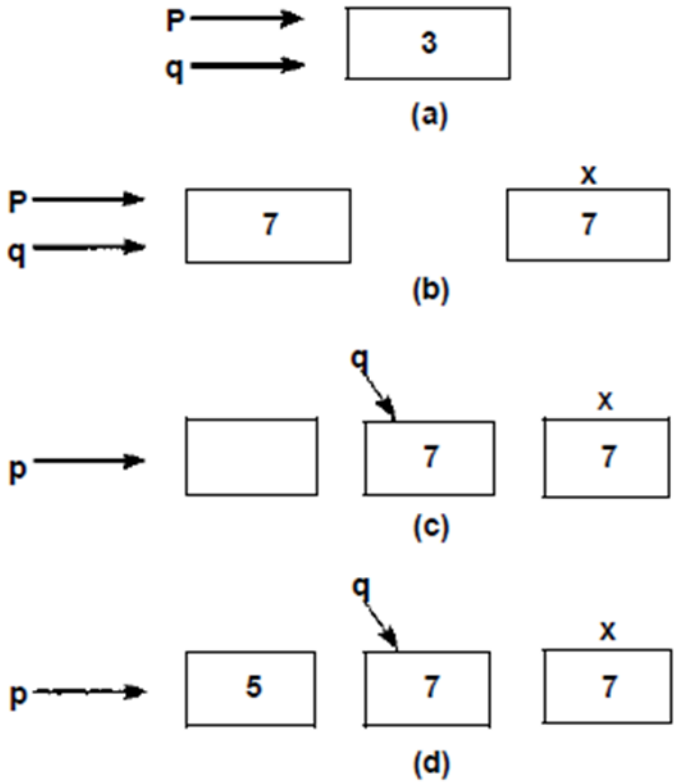
```
int *p, *q; int x;

p = (int *)malloc(sizeof(int));
*p = 3;
q = p;
printf("%d %d \n", *p, *q);
x = 7;
*q = x;
printf("%d %d\n", *p, *q);
p = (int *)malloc(sizeof(int));
*p = 5;
printf("%d %d\n", *p, *q);
```

Execução

```
int *p, *q; int x;

p = (int *)malloc(sizeof(int));
*p = 3;
q = p;
printf("%d %d \n", *p, *q);
x = 7;
*q = x;
printf("%d %d\n", *p, *q);
p = (int *)malloc(sizeof(int));
*p = 5;
printf("%d %d\n", *p, *q);
```



Exercício 1

- Utilizando alocação dinâmica criar um programa para criar dinamicamente um vetor de inteiros
- Solicitar ao usuário o tamanho do vetor
- Depois solicitar ao usuário os valores para preencher todas as posições do vetor.
- Em seguida, exibir os valores do vetor criado.
- Não precisa de menu e não esquecer de liberar o vetor alocado.

Exercício 2

- Alterar o código do primeiro exercício solicitando ao usuário, após a impressão dos valores, qual o tamanho para realocar o vetor.
- Ou seja, no caso de 10 posições, realocar para 20.
- Solicitar o preenchimento das posições criadas
- Imprimir novamente o vetor com todos os valores

Finalização