ESTRUTURAS DE DADOS I

Ciência da Computação

ATENÇÃO:

Conforme observação no Plano de Ensino disponível e publicitado a todos os alunos desta disciplina, comunico que não autorizo a gravação, filmagem, captação de imagens por meio de fotografias e congêneres, reprodução e ou divulgação dos materiais didáticos e de ensino de minha autoria produzidos, organizados e utilizados em sala de aula, sem a expressa autorização da minha parte. Dada ciência, sob a égide da Constituição Federal de 1988, e demais legislação vigente, em especial a Lei nº 9610/98, Art. 46, IV e de direito à imagem, desde já ficam todos sujeitos às devidas sanções administrativas e ou legais cabíveis.

Igor Yepes
SIAPE 2289347 - Professor EBTT
IFFar - Campus Frederico Westphalen

Alocação Dinâmica de Memória

Igor Yepes

Alocação dinâmica

A alocação dinâmica permite alocar memória durante a execução de um programa. Isso é bem interessante, pois permite que o espaço em memória seja alocado apenas quando necessário. Além disso, a alocação dinâmica permite aumentar ou até diminuir a quantidade de memória alocada.

sizeof - retorna o número de bytes para um determinado tipo de dados.
x = sizeof(int); //retorna 4 (em geral)

malloc - aloca um espaço de memória e retorna um ponteiro do tipo void para o início do espaço de memória alocado.

free - libera o espaço de memória alocado.

Alocação dinâmica de memória

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

int main (void) {
    int tam, *pVet;
    printf("Informe o tamanho do vetor: ");
    scanf("%d", &tam);

    pVet = (int*)malloc(tam * sizeof(int));

    for (int i = 0; i < tam; i++){
        printf("Informe o valor da posição %d: ", i);
        scanf("%d", &pVet[i]);
    }

    for (int i = 0; i < tam; i++){
        printf("Valor na posição %d = %d\n", i, pVet[i]);
    }

    free(pVet);
    pVet = NULL;
}
```

Libera novamente a memória para uso

limpeza do ponteiro - Opcional

```
Informe o tamanho do vetor: 5
Informe o valor da posição 0: 1
Informe o valor da posição 1: 2
Informe o valor da posição 2: 3
Informe o valor da posição 3: 4
Informe o valor da posição 4: 5
Valor na posição 0 = 1
Valor na posição 1 = 2
Valor na posição 2 = 3
Valor na posição 3 = 4
Valor na posição 4 = 5
```

tam * sizeof(int) = calcula o tamanho necessário a ser alocado para o tipo do vetor (int, float, ...)

malloc = reserva o espaço de memória necessário

Alocação dinâmica de memória (STRUCTS)

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

typedef struct funcionario {
    int cod;
    float salario;
}tFuncionario;

void alteraSalario(tFuncionario *pFunc, int tam) {
    for (int i = 0; i < tam; i++)
        pFunc[i].salario *= 1.3;
}

int main (void) {
    int tam;
    tFuncionario *pVet;

    printf("Digite o tamanho do cadastro: ");
    scanf("%d", &tam);

    pVet = (tFuncionario*)malloc(tam * sizeof(tFuncionario));

    for (int i = 0; i < tam; i++){
        printf("\nDigite o código: ");
        scanf("%d", &pVet[i].cod);
        printf("Digite o salário: ");
        scanf("%f", &pVet[i].salario);
    }

    alteraSalario(pVet, tam);

    printf("\n# Salários reajustados em 30%% #\n");
    for (int i = 0; i < tam; i++){
        printf("Código: %d  -  Salário: %.2f\n", pVet[i].cod, pVet[i].salario);
    }
    free(pVet);
    pVet = NULL;
}
```

```c
if (pVet == NULL) {
    printf("Memória insuficente\n");
    exit(1);
}
```

```
Digite o tamanho do cadastro: 5

Digite o código: 0001
Digite o salário: 2350.34

Digite o código: 0002
Digite o salário: 3760.56

Digite o código: 0003
Digite o salário: 4580.00

Digite o código: 0004
Digite o salário: 3500.00

Digite o código: 0005
Digite o salário: 1890.34

# Salários reajustados em 30% #
Código: 1  -  Salário: 3055.44
Código: 2  -  Salário: 4888.73
Código: 3  -  Salário: 5954.00
Código: 4  -  Salário: 4550.00
Código: 5  -  Salário: 2457.44
```

É interessante ter uma validação da alocação de memória, caso o sistema não tenha espaço disponível para alocar.

Alocação dinâmica de memória (MATRIZES)

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

int main(void){
    int i, j, lin, col, cont=0;
    int **mat; //vetor de vetores (** = ponteiro para ponteiro)

    printf("Digite a quantidade de linhas: ");
    scanf("%d", &lin);
    printf("Digite a quantidade de colunas: ");
    scanf("%d", &col);

    //aloca memória para o vetor de ponteiros (linhas)
    mat = (int**)malloc(lin * sizeof(int*));

    //aloca memória para cada vetor dentro do vetor principal (colunas)
    for(i = 0; i < lin; i++) {
        mat[i] = (int*)malloc(col * sizeof(int));
    }

    for (i = 0; i < lin; i++) {
        for (j = 0; j < col; j++) {
            mat[i][j] = cont++;
            printf("%3d ", mat[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < lin; i++) {
        free(mat[i]);
    }
    free(mat);
}
```

Acessa os dados da matriz da forma convencional
matriz [linha] [coluna]

Para liberar a memória, primeiro deve liberar os vetores internos e só depois o vetor principal

```
Digite a quantidade de linhas: 10
Digite a quantidade de colunas: 10
  0   1   2   3   4   5   6   7   8   9
 10  11  12  13  14  15  16  17  18  19
 20  21  22  23  24  25  26  27  28  29
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50  51  52  53  54  55  56  57  58  59
 60  61  62  63  64  65  66  67  68  69
 70  71  72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99
```