Aula prática 9

Alocação Dinâmica

Monitoria de Introdução à Programação

Ponteiros - Vetores

```
Char (*pString)[50];
```

- Declara que o conteúdo ao qual pString aponta é do tipo vetor de char de 50 posições, tornando pString um ponteiro para vetor de char de 50 posições.
- Como já foi dito, ponteiros guardam endereços. Vetores também. Portanto podemos acessar os valores do vetor usando ponteiros.

```
char string[20];
char* pchar = string;
```

Ponteiros - Matrizes

- Porém, para acessar matrizes através de ponteiros, temos que ter cuidado:

- Uma matriz é um espaço contínuo na memória, sendo acessado diferenciando somente um endereço:

```
int matriz[20][10];
matriz[i][j]; //isso*
(matriz + i*10 + j); //Equivale a isso
```

Ponteiros - Matrizes

- Pode-se também olhar a matriz como um vetor de vetores e portanto acessá-la usando ponteiro de ponteiro:

```
matriz[i][j]; //Isso
*( *(matriz + i) + j); //Equivale a isso
```

Alocação Dinâmica

- Ferramenta que possibilita a reserva de memória em tempo de execução.
- Possibilita a criação de programas mais eficientes, com um menor consumo de memória.
- Como toda ferramenta, a alocação dinâmica tem suas vantagens e desvantagens.

Alocação Dinâmica

- O primeiro passo para alocar vetores e matrizes de maneira dinâmica é aprendendo a utilizar as seguintes funções:
  - void* malloc(int size)
  - void* calloc(int n, int size)
  - void* realloc(void* pointer, int size)
  - void free(void* pointer)

Malloc

void* malloc(int size)

- Aloca na memória o número de bytes definido por size, e retorna o endereço do primeiro elemento desse espaço alocado.
- O retorno é do tipo void*, logo, é sempre necessário utilizar um cast ao se usar a função malloc.
- É recomendado sempre o uso do sizeof() para calcular a quantidade de espaço a ser alocada.

Malloc

void* malloc(int size)

- Caso não haja espaço suficiente na memória para alocar, a função retornará NULL
- Dessa maneira, podemos verificar durante a execução se o programa deve continuar ou finalizar usa execução.
- Esse comportamento é igual para todas as funções de

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Malloc

- Caso não haja espaço suficiente na memória para alocar, a função retornará NULL
- Dessa maneira, podemos verificar durante a execução se o programa deve continuar ou finalizar usa execução.
- Esse comportamento é igual para todas as funções de

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Malloc

- Caso não haja espaço suficiente na memória para alocar, a função retornará NULL
- Dessa maneira, podemos verificar durante a execução se o programa deve continuar ou finalizar usa execução.
- Esse comportamento é igual para todas as funções de

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Malloc

- Dessa maneira, podemos verificar durante a execução se o programa deve continuar ou finalizar usa execução.
- Esse comportamento é igual para todas as funções de

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Malloc

- Esse comportamento é igual para todas as funções de

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Malloc

```
int *p;
p = (int*) malloc(sizeof(int)*10);
```

Calloc

void* calloc(int n, int size)

- Aloca na memória o número de bytes definido por size multiplicado pelo valor de n, e retorna o endereço do primeiro elemento desse espaço alocado.
- A memória alocada é limpa no processo.
- Assim como o malloc, deve ser feito um cast no retorno.
- É recomendado sempre o uso do sizeof() para calcular a quantidade de espaço a ser alocada.

```
int *p;
p = (int*) calloc(10, sizeof(int));
```

Realloc

void* realloc(void* pointer, int size)

- Aloca na memória o número de bytes definido por size, e retorna o endereço do primeiro elemento desse espaço alocado.
- No processo, os elementos atuais da memória apontada por pointer são copiados
- Caso não haja memória suficiente, NULL é retornado e pointer permanece inalterado.
- Assim como no malloc e calloc, deve ser feito o cast do retorno.

Free

void free(void* pointer)

- Libera o espaço de memória alocado apontado por pointer que foi previamente alocado utilizando umas da funções vistas anteriormente.
- Caso não seja utilizada, o espaço alocado permanecerá bloqueado para outros usos.

Dúvidas?

Exercício 1

Faça um programa que crie dinamicamente um vetor de n elementos e passe esse vetor para uma função que vai ler os n elementos. Depois imprima o vetor.

O vetor não deve ser impresso na função e sim na main.

Antes de finalizar o programa, deve-se liberar a área de memória alocada.

Exercício 2

- Escreva um programa que solicita ao usuário um vetor de notas (números reais) e imprime a média aritmética das notas.

- **Apesar de não ser necessário utilize um vetor.**

- **O programa não deve limitar o tamanho do vetor.**

- **Não deve ocorrer desperdício de memória.**

- **Antes de finalizar o programa, deve-se liberar a área de memória alocada.**