Curso: Ciência da Computação

Disciplina: Estrutura de Dados 1

Professor: Clayton Zambon

centro universitário unifacvest

2. Introdução

2.2. Matrizes

2. Introdução
2.2. Matrizes

MATRIZES

O que é uma matriz?

Matriz é a uma estrutura de dados do tipo vetor com duas ou mais dimensões.

Os itens de uma matriz devem ser todos do mesmo tipo de dado.

Na prática, as matrizes formam tabelas na memória.

2. Introdução
2.2. Matrizes

MATRIZES

Matrizes01.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    //Declaração de uma Matriz 2x2
    //2 linhas e 2 colunas
    float matriz[2][2] = {
        {2,3}, {4,5}
    };
    int i, j;

    //Exibindo a Matriz
    for(i = 0; i < 2; i++)
    {
        for(j = 0; j < 2; j++)
        {
            printf("%.2f\t", matriz[i][j]);
        }
        printf("\n");
    }

    //Acessando o elemento da linha 1 e coluna 2
    printf("%.2f", matriz[0][1]);

    return 0;
}
```

2. Introdução
2.2. Matrizes

MATRIZES

- Como passar uma matriz para uma função?

Matrizes02.c

```c
#include <stdio.h>
//Veja que a funcao recebe a matriz
//sem o primeiro valor das linhas
//void f(float (*matriz)[2]) //Existe esta forma tbm
void f(float matriz[][2])
{
    int i, j;
    //Exibindo a Matriz
    for(i = 0; i < 2; i++)
    {
        for(j = 0; j < 2; j++)
        {
            printf("%.2f\t", matriz[i][j]);
        }
        printf("\n");
    }
}
int main(int argc, char** argv)
{
    //Declaração de uma Matriz 2x2
    //2 linhas e 2 colunas
    float matriz[2][2] = {
        {2,3}, {4,5}
    };
    //Chamamos a funcao passando a matriz criada
    //Veja que apenas enviamos o nome da matriz
    f(matriz);

    return 0;
}
```

2. Introdução
2.2. Matrizes

MATRIZES

- Existe outra forma de passar uma matriz por parâmetro:

– Void f(float **(*matriz)[2]) { … }**

MATRIZES

- Alocação dinâmica de matrizes:

- A linguagem C só permite alocar dinamicamente conjuntos unidimensionais;
- Para trabalha com matrizes alocadas dinamicamente, nós teremos que criar abstrações conceituais com vetores para representar conjuntos bidimensionais;
- Podemos armazenar a matriz em um vetor simples;
- Conceitualmente teremos um conjunto bidimensional mas de fato será um vetor unidimensional;

MATRIZES

- Alocação dinâmica de matrizes:

- Imaginemos que eu quero acessar o elemento matriz[i][j], onde:
  - v[k] → k = i * n + j
  - K: elemento que eu quero acessar;
  - I: número da linha onde o elemento está;
  - N: número de colunas da matriz;
  - J: número da coluna do elemento.

2. Introdução
2.2. Matrizes

MATRIZES

- Alocação dinâmica de matrizes:

			j		
			0	1	
i		0	1	2	
		1	3	4	matriz 2x2
	Vetor		1	2	3	4
			0	1	2	3
			v[k] → k = i * n + j			
			Pegando o elemento 3 da matriz fica:			
			K = 1 * 2 + 1			

2. Introdução
2.2. Matrizes

MATRIZES

- Alocação dinâmica de

Matrizes: Vetor simples

Matrizes03.c

```c
#include <stdio.h>
#include <stdlib.h>
#define LIN 2 //Constante global do numero de linhas
#define COL 2 //Constante global do numero de colunas

int main(int argc, char** argv)
{
    int *matriz;
    int i;

    matriz = (int*)malloc(LIN*COL*sizeof(int));

    if(matriz == NULL)
    {
        printf("Memoria insuficiente\n");
        exit(1);
    }

    //Preenchendo a matriz com valores
    for(i = 0; i < LIN*COL; i++)
    {
        matriz[i] = i;
    }

    //acessar o elemento da primeira linha e segunda coluna
    // k = i*n + j -> k = i*LIN*COL + j
    //int indice = 0*LIN*COL + 1;

    printf("%d\n", matriz[1]);

    return 0;
}
```

MATRIZES

- Alocação dinâmica de matrizes:

- Imaginemos que eu quero acessar o elemento matriz[i][j], onde:
  - v[k] → k = i * n + j
  - K: elemento que eu quero acessar;
  - I: número da linha onde o elemento está;
  - N: número de colunas da matriz;
  - J: número da coluna do elemento.

MATRIZES

- Alocação dinâmica de matrizes: Vetor de Ponteiros

- Cada linha da Matriz será representada por um Vetor independente;
- Será um vetor de ponteiros;
- Cada elemento irá armazenar o endereço do primeiro elemento de cada linha;

2. Introdução
2.2. Matrizes

MATRIZES

- Alocação dinâmica de matrizes: Vetor de Ponteiros.

Matrizes04.c

```c
#include <stdio.h>
#include <stdlib.h>
#define LIN 2 //Constante global do numero de linhas
#define COL 3 //Constante global do numero de colunas

int main(int argc, char** argv)
{
    int **matriz; //Vetor de Ponteiros
    int i;
    //Alocando o vetor de Ponteiros
    matriz = (int**)malloc(LIN * sizeof(int));

    //Preenchendo a matriz com valores
    for(i = 0; i < LIN; i++)
    {
        matriz[i] = (int*)malloc(COL * sizeof(int));
    }

    matriz[0][0] = 10;
    matriz[0][1] = 20;

    printf("%d\n", matriz[0][1]);

    //Liberando a memoria
    for(i = 0; i < LIN; i++)
    {
        free(matriz[i]);
    }

    free(matriz);

    return 0;
}
```

Referências

EDELWEISS, Nina; GALANTE, Renata. Estruturas de Dados. Porto Alegre, BOOKMAN, 2009.

HEINZLE, Roberto. Estruturas de Dados: implementações com C e Pascal. Blumenau, DIRETIVA, 2006.

TENENBAUM, Aron M. Estrutura de Dados usando C. São Paulo, Makron Books, 1995.

FORBELLONE, André Luiz Villar: EBERSPÄCHER, Henri Frederico. Lógica de Programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo, PRENTICE HALL, 2005.

KOFFMAN, Elliot B.; WOLFGANG, Paul A. T. Objetos, abstração, estruturas de dados e projeto usando C++. Rio de Janeiro, LTC, 2008.

PEREIRA, Silvio do lago. Estruturas de dados fundamentais: conceitos e aplicações. São Paulo, Érica, 1996.

VILLAS, Marcos Viana et al. Estruturas de dados – Conceitos e técnicas de implementação. Rio de Janeiro, Campus, 1993.

VELOSO, Paulo et al. Estrutura de dados. Rio de Janeiro, Campus, 1996.

Obrigado!