

UNIVERSIDADE FEDERAL DO AMAZONAS

PROGRAMAÇÃO EM TEMPO REAL
LABORATÓRIO 2

EDSON NASCIMENTO SILVA NETO - 21350694
EVALDO PATRIK DOS SANTOS CARDOSO - 21453640

Manaus - AM
Setembro de 2019

Sumário

1. Objetivos.....	2
2. Introdução Teórica.....	2
2.1. <i>Abstract Data type</i>	2
2.2. <i>Application Programming Interface</i>	2
3. Ambiente.....	2
3.1. matrix.h	2
3.2. matrix.c	3
3.3. lab2.c	3
3.4. Makefile.mk	3
4. Conclusão.....	3
5. Referências.....	3
6. Apêndice 1 - matrix.h.....	4
7. Apêndice 2 - matrix.c.....	5
8. Apêndice 3 - lab2.c.....	9
9. Apêndice 4 - Makefile.mk.....	10

1 . Objetivos

Este relatório descreve o trabalho de laboratório 2, que tem por objetivo a criação de uma ADT (*Abstract Data Type*) para manipulação de matrizes, uma API de manipulação de matrizes e um programa em C para teste das funções desenvolvidas.

2 . Introdução Teórica

2.1. *Abstract Data Types*

Abstract Data type (ADT) é um tipo (ou classe) para objetos cujo comportamento é definido por um conjunto de valores e um conjunto de operações.

A definição de ADT somente menciona quais operações devem ser realizadas, mas não como estas devem ser implementadas. Não especifica como os dados serão organizados na memória e quais algoritmos serão usados para implementar as operações. É chamado “abstrato” (*abstract*) porque dá uma visão independente de implementação. O processo de fornecer somente o essencial e esconder o detalhes é conhecido como **abstração**.

2.2. *Application Programming Interface*

Uma *Application Programming Interface* (API) é uma **interface** ou **protocolo de comunicação** entre um cliente e um servidor, feito para simplificar a construção do software cliente. Na construção de aplicações, uma API simplifica a programação **abstraindo** a implementação subjacente e somente expondo objetos ou ações que o desenvolvedor precisa.

Uma API geralmente está ligada a uma **biblioteca** de software. A API descreve e prescreve o “comportamento esperado” (a especificação) enquanto a biblioteca é a implementação em si do conjunto de regras.

3 . Ambiente

Descrição da estrutura da estrutura de diretórios utilizada no trabalho, a descrição dos códigos se encontra nos apêndices.

3.1. **matrix.h**

Arquivo de extensão .h onde estão descritas as especificações do ADT de matriz, além de todas as especificações das funções de manipulação de matrizes, sendo esta a API de nosso trabalho.

3.2. matrix.c

Arquivo de extensão .c onde a implementação de todas as regras especificadas em matrix.h se encontra, sendo esta a biblioteca do trabalho.

3.3. lab2.c

Arquivo de extensão .c a função *main()* se encontra, onde as as dependências e bibliotecas são importadas e utilizadas.

3.4. Makefile.mk

Arquivo de extensão .mk onde estão nossos comandos de *build*, execução e limpeza do projeto.


4. Conclusão

O objetivo do trabalho era implementar um ADT, uma interface de programação de aplicativo e biblioteca correspondente. O presente trabalho oferece a oportunidade do aprendizado de padrões de programação e desenvolvimento, tais como a correta utilização de *headers*, que permitem a estruturação de projetos em C e ADTs, mostrando a correta utilização de tipos abstratos.

5. Referências

1. https://en.wikipedia.org/wiki/Application_programming_interface#Libraries_and_frameworks. Acesso em 28/09/2019
2. <https://www.geeksforgeeks.org/abstract-data-types/>. Acesso em 28/09/2019

6. Apêndice 1 - matrix.h

 matrix.h

```
1  #ifndef _MATRIX_H
2  #define _MATRIX_H
3
4  #define M_MAX 10 // representa o num max de linhas
5  #define N_MAX 10 // representa o num max de colunas
6  #define MAX_NAME_SIZE 20 // tam maximo de caracteres no nome
7
8  // ADT da Matriz
9  typedef struct{
10     char name[MAX_NAME_SIZE];
11     int m; // numero de linhas
12     int n; // numero de colunas
13     double values[M_MAX][N_MAX]; // valores das linhas e colunas
14 } Matrix;
15
16 // printa os elementos da matriz na tela.
17 void matrix_print(Matrix mat);
18
19 // cria uma matriz de m linhas e n colunas, recebendo como parâmetros
20 // os elementos da ADT e um array de números (elements) de tamanho m*n
21 // para serem inseridos na matriz.
22 Matrix create_matrix(char name[MAX_NAME_SIZE], int m, int n, int elements[N_MAX*M_MAX]);
23
24 // cria uma matriz de m linhas e n colunas em que todos os elementos são 0.
25 Matrix matrix_zeros(char nome[MAX_NAME_SIZE], int m, int n);
26
27 // soma duas matrizes, se estas possuírem o mesmo tamanho.
28 Matrix matrix_add(Matrix mat1, Matrix mat2);
29
30 // subtrai duas matrizes, se estas possuírem o mesmo tamanho.
31 Matrix matrix_sub(Matrix mat1, Matrix mat2);
32
33 // multiplica das matrizes, se estas forem de tamanho
34 // Amxn e Bnxb , numero de colunas da primeira matriz igual
35 // ao numero de linhas da segunda matriz.
36 Matrix matrix_mult(Matrix mat1, Matrix mat2);
37
38 // retorna a transposta da matriz recebida.
39 Matrix matrix_transp(Matrix mat);
40
41 // multiplica uma matriz por um escalar.
42 Matrix matrix_scalar_mult(int scalar, Matrix mat);
43
44
45
46
47 #endif
```

7. Apêndice 2 - matrix.c

```
#include "matrix.h"
#include <stdio.h>
#include <string.h>

// realiza o print da matriz, função roubada do professor
void matrix_print(Matrix mat){

    printf("Matrix: %s\n",mat.name);

    for(int i=0; i < mat.m; i++){
        printf("\n" );
        for(int j=0; j < mat.n; j++){

            if(j == 0){

                printf("| ");
                printf("%f ", mat.values[i][j]);

            }else if(j == mat.n-1){

                printf("%f ", mat.values[i][j]);
                printf("|");

            }else{

                printf("%f ", mat.values[i][j]);
            }
        }
        printf("\n\n");
    }

    // cria uma matriz de zeros, outra função desavergonhadamente
    // plagiada do prof.
    Matrix matrix_zeros(char name[MAX_NAME_SIZE], int m, int n){
        Matrix mat;

        strncpy(mat.name, name, MAX_NAME_SIZE);
        mat.m = m;
        mat.n = n;

        for(int i=0; i < m; i++){
            for(int j=0; j < n; j++){
                mat.values[i][j] = 0;
            }
        }

        return mat;
    }

    // criação de matrizes
```

```

Matrix create_matrix(char name[MAX_NAME_SIZE], int m, int n, int
elements[N_MAX*M_MAX]){
    Matrix mat = matrix_zeros(name, m, n);

    int el = 0;
    for(int i=0; i < m; i++){
        for(int j=0; j < n; j++){
            mat.values[i][j] = elements[el];
            el += 1;
        }
    }

    return mat;
}

// adiç o de matrizes
Matrix matrix_add(Matrix mat1, Matrix mat2){

    Matrix mat3 = matrix_zeros("resultante", mat1.m, mat1.n);

    if(mat1.m != mat2.m || mat1.n != mat2.n){
        printf("Matrizes tem tamanhos diferentes\n");
        printf("operacao de soma nao pode ser realizada.\n");
        return mat3;
    }else {

        for(int i=0; i < mat1.m; i++){
            for(int j=0; j < mat1.n; j++){
                mat3.values[i][j] = mat1.values[i][j] + mat2.values[i][j];
            }
        }

        return mat3;
    }

}

// subtra o de matrizes
Matrix matrix_sub(Matrix mat1, Matrix mat2){
    Matrix mat3 = matrix_zeros("resultante", mat1.m, mat1.n);

    if(mat1.m != mat2.m || mat1.n != mat2.n){
        printf("Matrizes tem tamanhos diferentes\n");
        printf("operacao de subtracao nao pode ser realizada.\n");
        return mat3;
    }else {

        for(int i=0; i < mat1.m; i++){
            for(int j=0; j < mat1.n; j++){
                mat3.values[i][j] = mat1.values[i][j] - mat2.values[i][j];
            }
        }

    }
}

```

```

    }
    return mat3;

}

// multiplicação de matrizes
Matrix matrix_mult(Matrix mat1, Matrix mat2){
    Matrix mat3 = matrix_zeros("resultante", mat1.m, mat1.n);

    if(mat1.n != mat2.m){
        printf("Matrizes tem tamanhos incompatíveis de linhas e colunas\n");
        printf("operacao de multiplicacao nao pode ser realizada.\n");
        return mat3;
    }else {

        /*
        na multiplicacao, o numero de colunas da primeira matriz deve ser igual
        ao numero de linhas da segunda. A matriz resultante possuirá o numero de linhas
        da primeira matriz e o numero de colunas da segunda.
        */
        int sum = 0;
        for (int c = 0; c < mat1.m; c++) { // 1o loop - linhas da 1a matriz
            for (int d = 0; d < mat2.n; d++) { // 2o loop - colunas da 2a matriz
                for (int k = 0; k < mat2.m; k++) { // 3o loop - linhas da 2a matriz
                    sum = sum + mat1.values[c][k]*mat2.values[k][d];
                }

                mat3.values[c][d] = sum;
                sum = 0;
            }
        }
        return mat3;
    }

}

// calculo da matriz transposta
Matrix matrix_transp(Matrix mat){

    Matrix res = matrix_zeros("resultante", mat.m, mat.n);

    int i,j=0;
    for(i=0;i<mat.m;i++){
        for(j=0;j<mat.n;j++){
            res.values[i][j]=mat.values[j][i];
        }
    }

    return res;

}

// multiplicacao escalar x matriz

```



```
Matrix matrix_scalar_mult(int scalar, Matrix mat){  
  
    for(int i=0; i < mat.m; i++){  
        for(int j=0; i < mat.n; i++){  
            mat.values[i][j] = mat.values[i][j]*scalar;  
        }  
    }  
  
    return mat;  
  
}
```

8. Apêndice 3 - lab2.c

lab2.c

```
1  #include <stdio.h>
2  #include "matrix.h"
3
4  int main(int argc, char const *argv[]) {
5
6
7      printf("Criando uma matriz de zeros\n");
8      Matrix mat1 = matrix_zeros("matriz 1", 5, 5);
9      matrix_print(mat1);
10     printf("Criando outras duas matrizes \n");
11     int values[] = {1,2,3,4};
12     int values2[] = {6,7,8,9};
13     Matrix mat2 = create_matrix("matriz 2", 2, 2, values);
14     Matrix mat3 = create_matrix("matriz 3", 2, 2, values2);
15     matrix_print(mat2);
16     matrix_print(mat3);
17     printf("Somando as duas matrizes\n");
18     Matrix mat4 = matrix_add(mat2, mat3);
19     matrix_print(mat4);
20     printf("Subtraindo as duas matrizes\n");
21     mat4 = matrix_sub(mat2, mat3);
22     matrix_print(mat4);
23     printf("Multiplicando as duas matrizes\n");
24     mat4 = matrix_mult(mat2, mat3);
25     matrix_print(mat4);
26     printf("Multiplicando matriz 2 por escalar 2\n");
27     mat4 = matrix_scalar_mult(2, mat2);
28     matrix_print(mat4);
29     printf("Transposta da matriz 3\n");
30     mat4 = matrix_transp(mat3);
31     matrix_print(mat4);
32
33     return 0;
34 }
```

9. Apêndice 4 - Makefile

Makefile.mk

```
1  #   Universidade Federal do Amazonas - UFAM
2  #   Disciplina: Programação de Sistemas de Tempo Real - PTR
3  #   Author: Edson Nascimento Silva Neto - 21350694
4  #           Evaldo Patrik dos Santos Cardoso - 21453640
5  #
6
7  all: Lab2
8
9  Lab2: matrix.c lab2.c #arquivos de dependencia
10      clear
11      rm -rf Lab2
12      gcc -o Lab2 matrix.c lab2.c
13
14  exec: Lab2
15      ./Lab2
16
17  clean: #remove arquivo somasub
18      rm -rf Lab2
19      clear
```