Resposta da lista de fixação:

1. **Questão 1**
    a. **Resposta:**

Na computação, a estrutura de dados consiste no modo de armazenamento e organização de dados em um computador. Os dados armazenados não consistem apenas no local em que são armazenados, mas nas relações entre eles. A correta escolha da categoria dos dados acarretará na performance ou na velocidade que o computador irá encontrá-lo. São exemplos de estrutura de dados: pilhas, filas, listas, árvores, hashtables e grafos.

Os tipos abstratos de dados (TAD) são estruturas de dados capazes de representar os tipos de dados que não foram previstos nas linguagens de programação e geralmente são necessárias no desenvolvimento de aplicações mais complexas. São estruturas formadas por uma camada de dados e uma de operações.

```c
#ifndef MATRIZ_H
#define MATRIZ_H

typedef struct matriz Matriz;

Matriz * cria_matriz(int nl, int nc);
void libera_matriz(Matriz * mat);
int acessa_matriz(Matriz * mat, int i, int j);
int atribui_matriz(Matriz * mat, int i, int j, float  v);
int nlinhas(Matriz * mat);
int ncolunas(Matriz *mat);

#endif
```

Exemplo de TAD

    b. **Resposta:**

Array estático pode ser definido como sendo uma estrutura de dados primariamente unidimensional que armazena uma quantidade de dados pré-definida em nodos do mesmo tipo, na mesma variável e de forma sequencial na memória. Nessa estrutura, cada posição tem um índice referente à sua localização, na qual armazena um determinado dado.

Ex:

```c
int numeros[100];
int matriz[40][50];
```

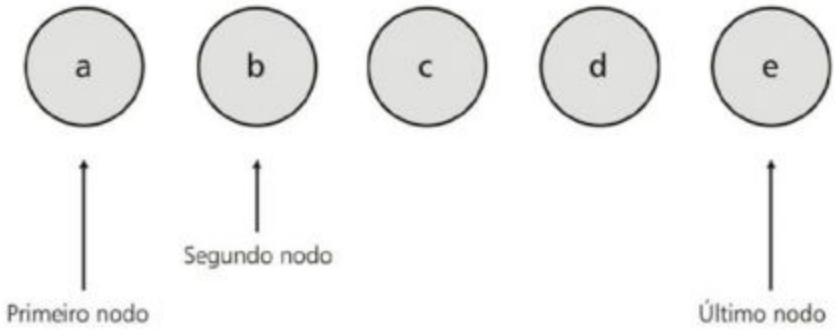
Array dinâmico são utilizados para relacionar itens que precisam ser manipulados em tempos de execução com dimensão indefinida. Durante o tempo de execução, é possível adicionar ou remover itens da estrutura.

Ex:

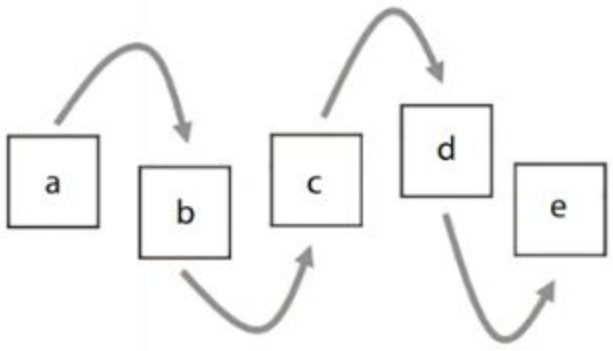
```
Matriz *mat = (Matriz*) malloc(sizeof (Matriz));
```

c. Resposta:

Uma lista pode ser definida como um conjunto de elementos do mesmo tipo, agrupados e identificados por um identificador único e separados entre si em nodos que ocupam um endereço específico na memória.



Lista encadeada constitui uma relação entre elementos interligados entre si, em que cada elemento é composto de uma estrutura que pode conter variáveis de diversos tipos de dados. Nesse caso, os elementos podem ser manipulados em tempo de execução. A estrutura de um elemento em uma lista encadeada é dividida em endereço de memória física do nodo, e duas partes que pertencem ao respectivo nodo, são elas o ponteiro para o próximo elemento e um espaço para armazenamento do dado do elemento. O endereço posterior ao último aponta para null.

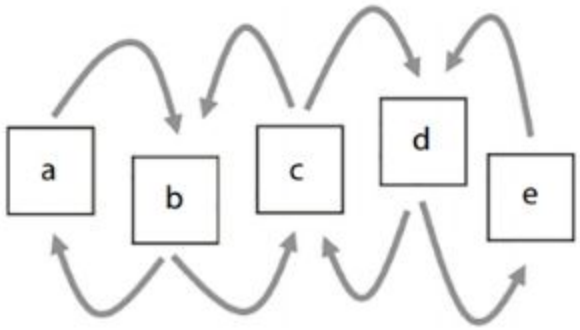


Representação de lista encadeada

```c
struct conjunto{
    int valor;
    struct conjunto* prox;
};
```

Estrutura de um elemento

Uma lista duplamente encadeada é uma sequência de itens que apontam para dois endereços, além de armazenar o conteúdo do próprio elemento. Os ponteiros correspondem aos endereços dos itens anterior e posterior na lista. Por conter esses recursos, o acesso aos elementos podem ocorrer em ambos os sentidos. Os endereços anterior ao primeiro elemento e posterior ao último apontam para null.

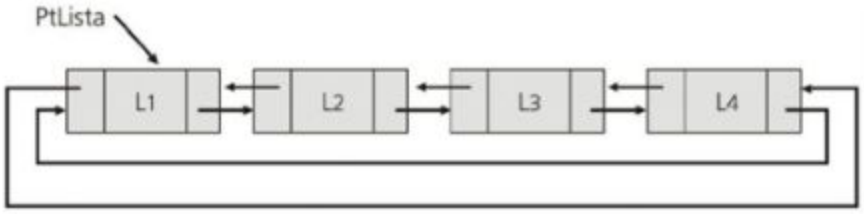


Representação de lista duplamente encadeada

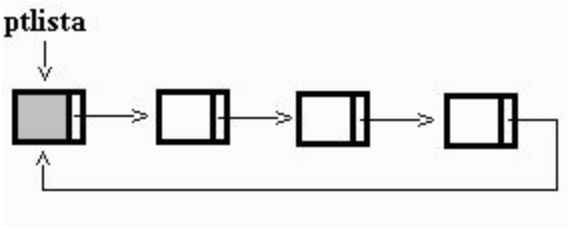
```c
struct conjunto{
    int valor;
    struct conjunto* prox;
    struct conjunto* ante;
};
```

Estrutura de um elemento

A lista circular é uma espécie da lista encadeada simples ou duplamente encadeada, em que no primeiro caso o último nodo irá apontar para o primeiro nodo da lista e no segundo caso o primeiro e o último nodo irão referenciar-se em seus respectivos ponteiros. Sendo assim, nas listas circulares, nunca chegaremos a uma posição a partir da qual não poderemos nos mover.



Representação de uma lista duplamente encadeada circular



Representação de uma lista simplesmente encadeada circular

Listas heterogêneas são estruturas que podem apresentar diversos tipos nos campos que armazenam os dados. Devem apresentar em sua estrutura um ponteiro para a próxima célula da lista, um ponteiro para a estrutura que contém a informação (tipo void*) e um identificador indicando qual objeto a célula armazena.

```
typedef struct exemplo_str{
    int tipo;
    void* Item;
    struct exemplo_str* Prox;
} Exemplo;
```

Estrutura de um elemento

2. **Questão 2**
    a. **Resposta**

```c
Conjunto * uniao(Conjunto **l1, Conjunto **l2){

    Conjunto * C = criar_conj_vazio(); // Criacao do conjunto vazio que sera retornado

    Conjunto * aux1 = *l1; // Criacao da copia do conjunto l1
    Conjunto * aux2 = *l2; // Criacao da copia do conjunto l2


    while (aux1 != NULL) //Pecorre o conjunto aux1
    {
        inserir(&C, aux1->valor); // Insere o valor em C

        aux1 = aux1->prox;

    }

    while (aux2 != NULL) //Pecorre o conjunto aux2
    {
        inserir(&C, aux2->valor); // Insere o valor em C

        aux2 = aux2->prox;
    }

    /*
        OBS: Na funcao inserir() ja existe um mecanismo que vizualiza se o
        valor ja pertence ao conjunto, portanto o tramento em relacao a isso eh feito na funcao.
    */

    return C; // Retorna C

}
```

b. Resposta:

```c
Conjunto * criar_conj_vazio(){
    // A criacao do conjunto vazio ocorre quando ele eh igual a NULL

    Conjunto * l = (Conjunto*) malloc(sizeof(Conjunto));
    if (l)
    {
        l = NULL; // Atribuicao ao conjunto
    }

    return l;
}
```

c. Resposta:

```c
int inserir(Conjunto **l1, int v){
    /*
        A insercao do elemento em um conjunto so ocorre se ele nao pertencer ao conjunto,
        uma vez que nao pode existir elementos repetidos.

        No caso, dessa funcao a insercao ocorre no inicio.
    */


    Conjunto * aux = create_elemento(v); // Criacao do elemento a ser add

    if(pertence(*l1,v)) // Verifica se o elemento ja existe no conjunto e retorna 0, porque nao eh add
        return 0;

    if(aux)
    {
        aux->prox = *l1; // Valor prox do elemento criado recebe o conjunto que ja existia
        *l1 = aux; // aux entao eh add no inicio

        return 1; // Retorna 1 como sucesso
    }

    return 0;
}
```

d. Resposta:

```c
int remover(Conjunto **l1, int v){
    Conjunto * ante = NULL;
    Conjunto * aux = *l1;
    int found = 0;

    while (aux != NULL) // Pecorre o conjunto aux a fim de procurar o valor que deseja remover
    {
        if (aux->valor == v) // Se for encontrado a variavel found = 1
        {
            found = 1;
            break;
        }

        ante = aux;
        aux = aux->prox;
    }

    if(found){ // Se o valor foi encontrado, entra nessa condicao
        if (ante == NULL) // Se entrar aqui, eh porque o valor escolhido eh o primeiro elemento
        {
            *l1 = aux->prox; // O conjunto deve receber o prox como valor de inicio
            free(aux); // Libera a memoria do valor escolhido
            return 1;
        }

        ante->prox = aux->prox; // O elemento posterior ao escolhido para remocao passa a ser o prox do antecessor
        free(aux); // Libera a memoria do valor escolhido
        return 1;
    }

    return 0; // retorna 0 se o elemento nao foi encontrado
}
```

e. Resposta:

```c
Conjunto * interseccao(Conjunto **l1, Conjunto **l2){
    Conjunto * C = criar_conj_vazio(); // Criacao do conjunto vazio que sera retornado

    Conjunto * aux1 = *l1; // Criacao da copia do conjunto l1
    Conjunto * aux2 = *l2; // Criacao da copia do conjunto l2

    /*
        Para que nao pecorra muitos elementos desnecessarios, vizualiza o tamanho
        e assim pega o de menor valor.
    */

    if (size(*l1) <= size(*l2)) // Se l1 for menor ou igual que l2
    {
        while (aux1 != NULL) // Pecorre o conjunto
        {
            if (pertence(aux2, aux1->valor)) // Se pertencer a ambos add em C
            {
                inserir(&C, aux1->valor);
            }

            aux1 = aux1->prox;

        }

        return C;
    }
    else // Se l1 for menor que l2
    {
        while (aux2 != NULL) // Pecorre o conjunto
        {
            if (pertence(aux1, aux2->valor)) // Se pertencer a ambos, add em C
            {
                inserir(&C, aux2->valor);
            }

            aux2 = aux2->prox;
        }

        return C;
    }

}
```

f. Resposta:

```c
Conjunto * diferenca(Conjunto **l1, Conjunto **l2){
    /*
        A funcao diferenca deve pecorrer o primeiro conjunto e analisar quais
        elementos nao pertencem ao segundo.
    */

    Conjunto * C = criar_conj_vazio(); // Criacao do conjunto vazio que sera retornado

    Conjunto * aux1 = *l1; // Criacao da copia do conjunto l1
    Conjunto * aux2 = *l2; // Criacao da copia do conjunto l2

    while (aux1 != NULL) // Pecorre o conjunto
    {
        if (!pertence(aux2, aux1->valor)) // Se nao pertencer ao segundo, add em C
        {
            inserir(&C, aux1->valor);
        }

        aux1 = aux1->prox;

    }

    return C; // Retorna C
}
```

g. Resposta:

```c
int pertence(Conjunto *l1, int v){
    if (conj_vazio(l1)) // Se l1 for vazio retorna 0
        return 0;

    while (l1 != NULL) // Pecorre o conjunto
    {
        if (l1->valor == v) // Se o elemento pertence ao conjunto, retorna 1 (verdadeiro)
            return 1;

        l1 = l1->prox;
    }

    return 0;
}
```

h. Resposta:

```c
int menor(Conjunto *l1){
    int menor = l1->valor; // Inicia o valor de menor com um elemento do conjunto

    while (l1 != NULL) // Pecorre o conjunto
    {
        if(l1->valor < menor) // Se o valor correspondente for menor, ele eh atribuido a variavel
            menor = l1->valor;

        l1 = l1->prox;
    }

    return menor; // Ao final de pecorrer o conjunto todo, retornar o menor valor
}
```

i. Resposta:

```c
int maior(Conjunto *l1){
    int maior = menor(l1); // Inicia o valor de maior com o menor valor do conjunto

    while (l1 != NULL) // Pecorre o conjunto
    {
        if(l1->valor > maior) // Se o valor correspondente for maior, ele eh atribuido a variavel
            maior = l1->valor;

        l1 = l1->prox;
    }

    return maior; // Ao final de pecorrer o conjunto todo, retornar o maior valor
}
```

j. Resposta:

```c
int iguais(Conjunto *l1, Conjunto *l2){
    /*
        Verifica se o valor da interseccao entre os dois conjuntos eh igual.
        Se for, quer dizer que os valores estao presentes nos dois conjuntos.
    */

    if(size(interseccao(&l1, &l2)) == size(l1) && size(interseccao(&l1, &l2)) == size(l2))
        return 1;

    return 0;
}
```

k. Resposta:

```c
int size(Conjunto *l1){ // Verifica tamanho do conjunto
    int counter = 0;

    while (l1 != NULL) // Pecorre o conjunto e incrementa o contador
    {
        counter++;
        l1 = l1->prox;

    }

    return counter; // Retorna o total de elementos
}
```

l. **Resposta:**

```c
int conj_vazio(Conjunto *l){
    return (l == NULL); // Se o conjunto l eh vazio retorna 1
}
```

m. **Resposta:**

```c
#include <stdio.h>
#include "conjunto.h"

int main(){

    Conjunto *conj1 = criar_conj_vazio(); // Criando conjunto vazio
    Conjunto *conj2 = criar_conj_vazio(); // Criando conjunto vazio
    Conjunto *conj_uniao;
    Conjunto *conj_interseccao;
    Conjunto *conj_diferenca;

    // Verificando se os conjuntos 1 e 2 sao vazios

    printf("%s\n", conj_vazio(conj1) ? "Conjunto vazio" : "Conjunto com elemento(s)");
    printf("%s\n", conj_vazio(conj2) ? "Conjunto vazio" : "Conjunto com elemento(s)");

    // Inserindo elementos no conjunto 1

    printf("\n");
    printf("%s\n", inserir(&conj1,8) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,8) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,3) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,6) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,2) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,9) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj1,7) ? "Valor inserido" : "Valor ja existe no conjunto");

    // Inserindo elementos no conjunto 2

    printf("\n");
    printf("%s\n", inserir(&conj2,1) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,7) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,3) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,6) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,5) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,11) ? "Valor inserido" : "Valor ja existe no conjunto");
    printf("%s\n", inserir(&conj2,7) ? "Valor inserido" : "Valor ja existe no conjunto");

    // Printando conjuntos
    printf("\n");
    printf("Conjunto 1: ");
    print(conj1);
    printf("\nConjunto 2: ");
    print(conj2);
    printf("\n");
```

```c
// Uniao conjunto 1 e 2

printf("\n");
conj_uniao = uniao(&conj1, &conj2); //chamada da funcao
printf("Conjunto uniao: ");
print(conj_uniao); //printando os elementos do conjunto retornado

// Interseccao conjunto 1 e 2

printf("\n");
conj_interseccao = interseccao(&conj1, &conj2); //chamada da funcao
printf("Conjunto interseccao: ");
print(conj_interseccao); //printando os elementos do conjunto retornado

// Diferenca de conjunto 1 e 2

printf("\n");
conj_diferenca = diferenca(&conj1, &conj2); //chamada da funcao
printf("Conjunto diferenca entre 1 e 2: ");
print(conj_diferenca); //printando os elementos do conjunto retornado

// Diferenca de conjunto 2 e 1

printf("\n");
conj_diferenca = diferenca(&conj2, &conj1); //chamada da funcao
printf("Conjunto diferenca entre 2 e 1: ");
print(conj_diferenca); //printando os elementos do conjunto retornado
printf("\n");

// Funcao pertence

printf("\n");
printf("%s\n", pertence(conj1, 5) ? "Pertence" : "Nao pertence");
printf("%s\n", pertence(conj1, 8) ? "Pertence" : "Nao pertence");
printf("%s\n", pertence(conj1, 9) ? "Pertence" : "Nao pertence");
printf("%s\n", pertence(conj2, 5) ? "Pertence" : "Nao pertence");
printf("%s\n", pertence(conj2, 1) ? "Pertence" : "Nao pertence");
printf("%s\n", pertence(conj2, 4) ? "Pertence" : "Nao pertence");

//Menor e maior

printf("\n");
printf("Maior valor do conjunto 1: %i\n", maior(conj1));
printf("Menor valor do conjunto 1: %i\n", menor(conj1));
printf("Maior valor do conjunto 2: %i\n", maior(conj2));
printf("Menor valor do conjunto 2: %i\n", menor(conj2));
```

```c
    // Funcao iguais

    printf("\n");
    printf("%s\n", iguais(conj1, conj2) ? "Os conjuntos sao iguais" : "Os conjuntos sao diferentes");

    // Tamanho dos conjuntos

    printf("\n");
    printf("Tamanho do conjunto 1: %i\n", size(conj1));
    printf("Tamanho do conjunto 2: %i\n", size(conj2));

    // Testando remocao

    printf("\n");
    printf("%s\n", remover(&conj1, 9) ? "Removido" : "Nao encontrado");
    printf("Tamanho do conjunto 1: %i\n", size(conj1));
    printf("%s\n", remover(&conj1, 20) ? "Removido" : "Nao encontrado");
    printf("Tamanho do conjunto 1: %i\n", size(conj1));
    printf("%s\n", remover(&conj2, 1) ? "Removido" : "Nao encontrado");
    printf("Tamanho do conjunto 2: %i\n", size(conj2));
    printf("%s\n", remover(&conj2, 14) ? "Removido" : "Nao encontrado");
    printf("Tamanho do conjunto 2: %i\n", size(conj2));

    return 0;
}
```

3. **Questão 3**
    a. **Resposta:**

```c
MATRIZ * cria_matriz(int n_linhas, int n_colunas){
    //Inicializacao da matriz com os valores enviados

    MATRIZ *mat = (MATRIZ*) malloc(sizeof(MATRIZ));
    mat->a_linhas = (PONT*) malloc(n_linhas*sizeof(PONT));

    if(mat){
        mat->n_linhas = n_linhas;
        mat->n_colunas = n_colunas;
        for (int i=0; i < n_linhas; i++) // Pecorre todas as linhas para inicializa-las com valor NULL
            mat->a_linhas[i] = NULL;
    }

    return mat;
}
```

    b. **Resposta:**

```c
void libera_matriz(MATRIZ * mat){
    free(mat); // Libera a matriz
}
```

    c. **Resposta:**

```c
int atribui_elemento(MATRIZ * mat, int lin, int col, float v){
    if (lin-1<0 || lin-1 >= mat->n_linhas || col-1<0 || col-1 >= mat->n_colunas) // Verifica se os valores repassados corresponde ao tamanho da matriz criada
        return 0;

    PONT ant = NULL;
    PONT atual = mat->a_linhas[lin-1]; // Seleciona a linha correspondente a enviada como parametro

    while (atual != NULL && atual->col < col-1) { // Pecorre os elementos ate o valor da coluna correspondente ser igual ao do parametro enviado
        ant = atual;
        atual = atual->prox;
    }

    if (atual != NULL && atual->col == col-1) { // Entra nesse if se o elemento onde quer add for diferente de NULL
        // Tratamento para valor igual a 0 que nao precisa ser alocado na memoria
        if (v == 0) {
            if (ant==NULL)
                mat->a_linhas[lin-1] = atual->prox;
            else
                ant->prox = atual->prox;

            free(atual); // Libera o espaco do elemento
        }
        else // Se o valor de v nao for 0, faz a atribuicao
            atual->v = v;
    }

    else if (v != 0) {
        PONT novo = (PONT) malloc(sizeof(ELEMENTO)); // Armazena espaco pro novo elemento

        // Atribui os valores as variaveis correspondentes
        novo->col = col-1;
        novo->v = v;
        novo->prox = atual;

        // Adiciona o elemento dependendo do valor do anterior a posicao desejada
        if (ant == NULL)
            mat->a_linhas[lin-1] = novo;

        else
            ant->prox = novo;
    }
    return 1; // Retorna 1 pra dizer que foi add
}
```

d. Resposta:

```c
float acessar_elemento(MATRIZ* mat, int lin, int col) {
    if (lin-1 < 0 || lin-1 >= mat->n_linhas || col-1 < 0 || col-1 >= mat->n_colunas) // Se o valor que foi repassado para a linha ou coluna nao corresponder a matriz retorna 0
        return 0;

    PONT atual = mat->a_linhas[lin-1]; // Seleciona a linha correspondente a enviada como parametro

    while (atual != NULL && atual->col < col-1) // Pecorre os elementos ate a coluna ser igual ao do paramento enviado
        atual = atual->prox;

    if (atual !=NULL && atual->col == col-1) // Verifica se eh o elemento que quer acessar
        return atual->v; // Retorna o valor do elemento

    return 0;
}
```

e. Resposta:

```c
int remover_elemento(MATRIZ * mat, int lin, int col){
    if (lin-1 < 0 || lin-1 >= mat->n_linhas || col-1 < 0 || col-1 >= mat->n_colunas) // Se o valor que foi repassado para a linha ou coluna nao corresponder a matriz retorna 0
        return 0;

    PONT atual = mat->a_linhas[lin-1]; // Seleciona a linha correspondente a enviada como parametro

    while (atual != NULL && atual->col < col-1) // Pecorre os elementos ate a coluna ser igual ao do paramento enviado
        atual = atual->prox;

    if (atual != NULL && atual->col == col-1) // Verifica se eh o elemento que quer excluir
    {
        free(atual); // Libera o espaco de memoria do mesmo
        return 1;
    }

    return 0;
}
```

f. Resposta:

```c
void print_matriz(MATRIZ * mat){
    int total_linhas = mat->n_linhas;
    int total_colunas = mat->n_colunas;

    for(int i = 0; i < total_linhas; i++) // Pecorre todas as linhas
    {
        for(int j = 0; j < total_colunas; j++) // Pecorre todas as colunas da linha correspondente
        {
            printf("%.2f  ", acessar_elemento(mat,i+1,j+1)); // Mostra o valor naquela posicao
        }
            printf("\n");
    }
}
```

g. Resposta:

```c
#include <stdio.h>
#include "matriz.h"

int main(){

    MATRIZ *matriz = cria_matriz(10,5); // Cria matriz com o total de 10 linhas e 5 colunas

    // Adicionando elemento a matriz

    /*
       A funcao adicionar foi adptada para a coluna e a linha que for enviada como paramento ser a
       correspondente na matriz. E nao ser uma a mais.
    */

    printf("\n");
    printf("%s\n", atribui_elemento(matriz,3,1,8) ? "Adicionado" : "Linha ou coluna inexistente");
    printf("%s\n", atribui_elemento(matriz,5,2,5) ? "Adicionado" : "Linha ou coluna inexistente");
    printf("%s\n", atribui_elemento(matriz,10,3,7) ? "Adicionado" : "Linha ou coluna inexistente");
    printf("%s\n", atribui_elemento(matriz,8,4,3) ? "Adicionado" : "Linha ou coluna inexistente");
    printf("%s\n", atribui_elemento(matriz,13,1,6) ? "Adicionado" : "Linha ou coluna inexistente");

    // Exibir matriz

    printf("\n");
    print_matriz(matriz);

    // Acessar elemento

    printf("\n");
    printf("Elemento da posicao: %.2f\n", acessar_elemento(matriz,8,4));
    printf("Elemento da posicao: %.2f\n", acessar_elemento(matriz,10,4));
    printf("Elemento da posicao: %.2f\n", acessar_elemento(matriz,3,1));

    // Remover elemento

    printf("\n");
    printf("%s\n", remover_elemento(matriz,3,1) ? "Removido" : "Linha ou coluna inexistente");
    printf("%s\n", remover_elemento(matriz,10,3) ? "Removido" : "Linha ou coluna inexistente");
    printf("%s\n", remover_elemento(matriz,13,1) ? "Removido" : "Linha ou coluna inexistente");

    printf("\n");
    print_matriz(matriz);

    // Libera matriz

    libera_matriz(matriz);

    return 0;
```