

```
/**
 * Arquivo: pontoTAD.h
 * Versão : 1.0
 * Data   : 2024-10-14 19:30
 * -----
 * Esta interface cria dois Tipos Abstratos de Dados, Ponto2D e Ponto3D, para
 * armazenar coordenadas de pontos no plano e no espaço. Também fornece funções
 * para criar e remover esses pontos, e calcular a distância entre 2 pontos.
 *
 * Prof.: Abrantes Araújo Silva Filho (Computação Raiz)
 *      www.computacaoraiz.com.br
 *      www.youtube.com.br/computacaoraiz
 *      github.com/computacaoraiz
 *      twitter.com/ComputacaoRaiz
 *      www.linkedin.com/company/computacaoraiz
 *      www.abrantes.pro.br
 *      github.com/abrantesasf
 */

/* Inicia boilerplate da interface: */

#ifndef _PONTOSTAD_H
#define _PONTOSTAD_H

/** Includes: */

/** Tipos Abstratos de Dados: */

/**
 * TIPO DE DADO: Ponto2D
 * -----
 * Este tipo de dado armazena as coordenadas X e Y de um ponto no plano,
 * considerando um sistema de coordenadas cartesiano. Para a criação de um
 * Ponto2D, ver a documentação da função "criar_Ponto2D" neste documento.
 */

typedef struct st_Ponto2D *Ponto2D;

/**
 * TIPO DE DADO: Ponto3D
 * -----
 * Este tipo de dado armazena as coordenadas X, Y e Z de um ponto no espaço,
 * considerando um sistema de coordenadas cartesiano. Para a criação de um
 * Ponto3D, ver a documentação da função "criar_Ponto3D" neste documento.
 */

typedef struct st_Ponto3D *Ponto3D;

/** Declarações de Subprogramas: */

/**
 * FUNÇÃO: criar_Ponto2D
 * Uso: criar_Ponto2D(x, y);
 * -----
 * Esta função recebe dois valores double, x e y, que correspondem às
 * coordenadas x (abscissa) e y (ordenada) de um ponto no plano cartesiano,
 * e retorna um ponteiro para um Ponto2D.
 *
 * Depois de criado esse ponto precisa ser apagado com o procedimento
 * "apagar_Ponto2D", antes da finalização do programa.
 */
```

```
*/

Ponto2D criar_Ponto2D (double x, double y);

/**
 * FUNÇÃO: criar_Ponto3D
 * Uso: criar_Ponto3D(x, y, z);
 * -----
 * Esta função recebe três valores double, x, y e z, que correspondem às
 * coordenadas x (abscissa), y (ordenada) e z (profundidade) de um ponto no
 * espaço cartesiano, e retorna um ponteiro para um Ponto3D.
 *
 * Depois de criado esse ponto precisa ser apagado com o procedimento
 * "apagar_Ponto3D", antes da finalização do programa.
 */

Ponto3D criar_Ponto3D (double x, double y, double z);

/**
 * PROCEDIMENTO: apagar_Ponto2D
 * Uso: apagar_Ponto2D(&P);
 * -----
 * Este procedimento recebe um PONTEIRO para um Ponto2D como argumento e libera
 * a memória alocada para esse Ponto2D, fazendo com que o Ponto2D aponte para
 * NULL para evitar dangling pointer.
 */

void apagar_Ponto2D (Ponto2D *P);

/**
 * PROCEDIMENTO: apagar_Ponto3D
 * Uso: apagar_Ponto3D(&P);
 * -----
 * Este procedimento recebe um PONTEIRO para um Ponto3D como argumento e libera
 * a memória alocada para esse Ponto3D, fazendo com que o Ponto3D aponte para
 * NULL para evitar dangling pointer.
 */

void apagar_Ponto3D (Ponto3D *P);

/**
 * FUNÇÃO: Ponto2D_getX
 * Uso: Ponto2D_getX(P);
 * -----
 * Esta função recebe como argumento um Ponto2D e retorna a coordenada X
 * (abscissa) deste ponto, como um valor double.
 */

double Ponto2D_getX (Ponto2D P);

/**
 * FUNÇÃO: Ponto2D_getY
 * Uso: Ponto2D_getY(P);
 * -----
 * Esta função recebe como argumento um Ponto2D e retorna a coordenada Y
 * (ordenada) deste ponto, como um valor double.
 */

double Ponto2D_getY (Ponto2D P);
```

```
/**
 * FUNÇÃO: Ponto3D_getX
 * Uso: Ponto3D_getX(P);
 * -----
 * Esta função recebe como argumento um Ponto3D e retorna a coordenada X
 * (abscissa) deste ponto, como um valor double.
 */

double Ponto3D_getX (Ponto3D P);

/**
 * FUNÇÃO: Ponto3d_getY
 * Uso: Ponto3D_getY(P);
 * -----
 * Esta função recebe como argumento um Ponto3D e retorna a coordenada Y
 * (ordenada) deste ponto, como um valor double.
 */

double Ponto3D_getY (Ponto3D P);

/**
 * FUNÇÃO: Ponto2D_getZ
 * Uso: Ponto2D_getZ(P);
 * -----
 * Esta função recebe como argumento um Ponto2D e retorna a coordenada Z
 * (profundidade) deste ponto, como um valor double.
 */

double Ponto3D_getZ (Ponto3D P);

/**
 * PROCEDIMENTO: Ponto2D_setX
 * Uso: Ponto2D_setX(Ponto2D P, double x);
 * -----
 * Esta função recebe como argumentos um Ponto2D e um valor double
 * correspondendo a uma coordenada X (abscissa), e atribui esse valor como a
 * nova abscissa do Ponto2D.
 */

void Ponto2D_setX (Ponto2D P, double x);

/**
 * PROCEDIMENTO: Ponto2D_setY
 * Uso: Ponto2D_setY(Ponto2D P, double y);
 * -----
 * Esta função recebe como argumentos um Ponto2D e um valor double
 * correspondendo a uma coordenada Y (ordenada), e atribui esse valor como a
 * nova ordenada do Ponto2D.
 */

void Ponto2D_setY (Ponto2D P, double y);

/**
 * PROCEDIMENTO: Ponto3D_setX
 * Uso: Ponto3D_setX(Ponto3D P, double x);
 * -----
 * Esta função recebe como argumentos um Ponto3D e um valor double
 * correspondendo a uma coordenada X (abscissa), e atribui esse valor como a
 * nova abscissa do Ponto3D.
 */
```

```
void Ponto3D_setX (Ponto3D P, double x);

/**
 * PROCEDIMENTO: Ponto3D_setY
 * Uso: Ponto3D_setY(Ponto3D P, double y);
 * -----
 * Esta função recebe como argumentos um Ponto3D e um valor double
 * correspondendo a uma coordenada Y (ordenada), e atribui esse valor como a
 * nova ordenada do Ponto3D.
 */

void Ponto3D_setY (Ponto3D P, double y);

/**
 * PROCEDIMENTO: Ponto3D_setZ
 * Uso: Ponto3D_setZ(Ponto3D P, double z);
 * -----
 * Esta função recebe como argumentos um Ponto3D e um valor double
 * correspondendo a uma coordenada Z (profundidade), e atribui esse valor como a
 * nova profundidade do Ponto3D.
 */

void Ponto3D_setZ (Ponto3D P, double z);

/**
 * FUNÇÃO: euclidiana_2d
 * Uso: euclidiana_2d(Ponto2D, Ponto2D);
 * -----
 * A função recebe como argumentos dois Ponto2D e retorna a distância
 * Euclidiana entre esses dois pontos, no plano.
 */

double euclidiana_2d (Ponto2D P, Ponto2D Q);

/**
 * FUNÇÃO: euclidiana_3d
 * Uso: euclidiana_3d(Ponto3D, Ponto3D);
 * -----
 * A função recebe como argumentos dois Ponto3D e retorna a distância
 * Euclidiana entre esses dois pontos, no espaço.
 */

double euclidiana_3d (Ponto3D P, Ponto3D Q);

/* Finaliza boilerplate: */

#endif
```