

```
/**
 * Arquivo: pontoTAD.c
 * Versão : 1.0
 * Data   : 2024-10-14 19:44
 * -----
 * Este arquivo implementa a interface pontoTAD.h.
 *
 * 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
 */

/** Includes: */

#include <math.h>
#include "pontoTAD.h"
#include <stdio.h>
#include <stdlib.h>

/** Tipos Abstratos de Dados: */

/**
 * TIPO DE DADO: Ponto2D
 * -----
 * Implementação do tipo de dado Ponto2D, que armazena as coordenadas de um
 * ponto no plano, considerando um sistema de coordenadas cartesiano.
 */

struct st_Ponto2D
{
    double coordX;
    double coordY;
};

/**
 * TIPO DE DADO: Ponto3D
 * -----
 * Implementação do tipo de dado Ponto3D, que armazena as coordenadas de um
 * ponto no espaço, considerando um sistema de coordenadas cartesiano.
 */

struct st_Ponto3D
{
    double coordX;
    double coordY;
    double coordZ;
};

/** Definições de Subprogramas: */

/**
 * FUNÇÃO: criar_Ponto2D
 * Uso: criar_Ponto2D(x, y);
 * -----
 * Um Ponto2D é um ponteiro para uma struct st_Ponto2D e, portanto, essa função
```

```
* retorna um ponteiro para essa struct (indiretamente, através do retorno de um
* Ponto2D). As coordenadas são mantidas em variáveis double independentes.
*/
```

```
Ponto2D criar_Ponto2D (double x, double y)
{
    Ponto2D T = malloc(sizeof(struct st_Ponto2D));
    if (T == NULL)
    {
        printf("Não foi possível criar o Ponto2D!\n");
        exit(1);
    }

    T->coordX = x;
    T->coordY = y;

    return T;
}

/**
 * FUNÇÃO: criar_Ponto3D
 * Uso: criar_Ponto3D(x, y, z);
 * -----
 * Um Ponto3D é um ponteiro para uma struct st_Ponto3D e, portanto, essa função
 * retorna um ponteiro para essa struct (indiretamente, através do retorno de um
 * Ponto3D). As coordenadas são mantidas em variáveis double independentes.
 */

Ponto3D criar_Ponto3D (double x, double y, double z)
{
    Ponto3D T = malloc(sizeof(struct st_Ponto3D));
    if (T == NULL)
    {
        printf("Não foi possível criar o Ponto3D!\n");
        exit(1);
    }

    T->coordX = x;
    T->coordY = y;
    T->coordZ = z;

    return T;
}

/**
 * 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. É necessário utilizar um PONTEIRO para um
 * Ponto2D pois, caso contrário, não seria possível atribuir o valor NULL ao
 * Ponto2D que está sendo desalocado (ponteiros são passados por valor).
 */
```

```
void apagar_Ponto2D (Ponto2D *P)
{
    if (*P != NULL)
    {
        free(*P);
    }
}
```

```
        *P = NULL;
    }
}

/**
 * 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. É necessário utilizar um PONTEIRO para um
 * Ponto3D pois, caso contrário, não seria possível atribuir o valor NULL ao
 * Ponto3D que está sendo desalocado (ponteiros são passados por valor).
 */

void apagar_Ponto3D (Ponto3D *P)
{
    if (*P != NULL)
    {
        free(*P);
        *P = NULL;
    }
}

/**
 * FUNÇÃO: Ponto2D_getX
 * Uso: Ponto2D_getX(P);
 * -----
 * Verifica se o Ponto2D não é nulo e retorna a coordenada X.
 */

double Ponto2D_getX (Ponto2D P)
{
    if (P == NULL)
    {
        printf("Erro ao obter a coordenada X do ponto especificado.\n");
        exit(1);
    }
    return P->coordX;
}

/**
 * FUNÇÃO: Ponto2D_getY
 * Uso: Ponto2D_getY(P);
 * -----
 * Verifica se o Ponto2D não é nulo e retorna a coordenada Y.
 */

double Ponto2D_getY (Ponto2D P)
{
    if (P == NULL)
    {
        printf("Erro ao obter a coordenada Y do ponto especificado.\n");
        exit(1);
    }
    return P->coordY;
}

/**
 * FUNÇÃO: Ponto3D_getX
```

```
* Uso: Ponto3D_getX(P);
* -----
* Verifica se o Ponto3D não é nulo e retorna a coordenada X.
*/

double Ponto3D_getX (Ponto3D P)
{
    if (P == NULL)
    {
        printf("Erro ao obter a coordenada X do ponto especificado.\n");
        exit(1);
    }
    return P->coordX;
}

/**
* FUNÇÃO: Ponto3d_getY
* Uso: Ponto3D_getY(P);
* -----
* Verifica se o Ponto3D não é nulo e retorna a coordenada Y.
*/

double Ponto3D_getY (Ponto3D P)
{
    if (P == NULL)
    {
        printf("Erro ao obter a coordenada Y do ponto especificado.\n");
        exit(1);
    }
    return P->coordY;
}

/**
* FUNÇÃO: Ponto2D_getZ
* Uso: Ponto2D_getZ(P);
* -----
* Verifica se o Ponto3D não é nulo e retorna a coordenada Z.
*/

double Ponto3D_getZ (Ponto3D P)
{
    if (P == NULL)
    {
        printf("Erro ao obter a coordenada Z do ponto especificado.\n");
        exit(1);
    }
    return P->coordZ;
}

/**
* PROCEDIMENTO: Ponto2D_setX
* Uso: Ponto2D_setX(Ponto2D P, double x);
* -----
* Verifica se o Ponto2D não é nulo e atribui coordenada X.
*/

void Ponto2D_setX (Ponto2D P, double x)
{
    if (P == NULL)
    {
```

```
        printf("Erro ao atribuir a coordenada X ao ponto especificado.\n");
        exit(1);
    }
    P->coordX = x;
}

/**
 * PROCEDIMENTO: Ponto2D_setY
 * Uso: Ponto2D_setY(Ponto2D P, double y);
 * -----
 * Verifica se o Ponto2D não é nulo e atribui coordenada Y.
 */

void Ponto2D_setY (Ponto2D P, double y)
{
    if (P == NULL)
    {
        printf("Erro ao atribuir a coordenada Y ao ponto especificado.\n");
        exit(1);
    }
    P->coordY = y;
}

/**
 * PROCEDIMENTO: Ponto3D_setX
 * Uso: Ponto3D_setX(Ponto3D P, double x);
 * -----
 * Verifica se o Ponto3D não é nulo e atribui coordenada X.
 */

void Ponto3D_setX (Ponto3D P, double x)
{
    if (P == NULL)
    {
        printf("Erro ao atribuir a coordenada X ao ponto especificado.\n");
        exit(1);
    }
    P->coordX = x;
}

/**
 * PROCEDIMENTO: Ponto3D_setY
 * Uso: Ponto3D_setY(Ponto3D P, double y);
 * -----
 * Verifica se o Ponto3D não é nulo e atribui coordenada Y.
 */

void Ponto3D_setY (Ponto3D P, double y)
{
    if (P == NULL)
    {
        printf("Erro ao atribuir a coordenada Y ao ponto especificado.\n");
        exit(1);
    }
    P->coordY = y;
}

/**
 * PROCEDIMENTO: Ponto3D_setZ
 * Uso: Ponto3D_setZ(Ponto3D P, double z);
```

```
* -----
* Verifica se o Ponto3D não é nulo e atribui coordenada Z.
*/

void Ponto3D_setZ (Ponto3D P, double z)
{
    if (P == NULL)
    {
        printf("Erro ao atribuir a coordenada Z ao ponto especificado.\n");
        exit(1);
    }
    P->coordZ = 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)
{
    if (P == NULL || Q == NULL)
    {
        printf("Erro ao calcular distância Euclidiana no plano.\n");
        exit(1);
    }
    return sqrt(pow((P->coordX - Q->coordX), 2.0) +
                pow((P->coordY - Q->coordY), 2.0));
}

/**
* 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)
{
    if (P == NULL || Q == NULL)
    {
        printf("Erro ao calcular a distancia Euclidiana no espaço.\n");
    }
    return sqrt(pow((P->coordX - Q->coordX), 2.0) +
                pow((P->coordY - Q->coordY), 2.0) +
                pow((P->coordZ - Q->coordZ), 2.0));
}
```