**LABORATÓRIO DE PROGRAMAÇÃO AVANÇADA**
**SEXTO TRABALHO PRÁTICO**
**- - TIPOS ABSTRATOS DE DADOS - -**

# DESCRIÇÃO

Tipo Abstrato de Dados (TAD) é uma abstração muito parecida com as classes de objetos da POO. Em C, um tipo abstrato de dados pode ser implementado por módulos. No caso, se um módulo contém a definição de um tipo de dados + operação sobre dados desse tipo + restrições sobre dados e operações, então tempos um Tipo Abstrato de Dados. Abstrato significa "esquecida a forma de implementar". Um TAD é descrito pela finalidade do tipo e de suas operações, e não pela forma como está implementado. Criando um tipo abstrato, podemos "esconder" a estratégia de implementação. Quem usa o tipo abstrato não precisa conhecer a forma como ele é implementado.# Isto facilita a manutenção e o re-uso de códigos.

Um programa em C pode conter um ou mais módulos, onde cada módulo vai conter dados e funções que representam apenas parte da implementação de um programa completo. Cada um desses módulos deve ser compilado separadamente, gerando um arquivo objeto (geralmente um arquivo com extensão ".o").

Para evitar a inclusão dos cabeçalhos das funções no Programa Principal (o que chama os módulos), todo módulo de funções C costuma ter associado a ele um arquivo que contém apenas os cabeçalhos das funções oferecidas pelo módulo e, eventualmente, os tipos de dados que ele exporte (typedef's, struct's, etc).

Usualmente, esse arquivo de cabeçalhos segue o mesmo nome do módulo ao qual está associado, só que com a extensão ".h".

# EXEMPLO – TAD PONTO

Vamos considerar a criação de um tipo de dado para representar um ponto no R2. Para isso, devemos definir um TAD Ponto. As operações são as seguintes:

- `cria`: operação que cria um ponto com coordenadas x e y;
- `libera`: operação que libera a memória alocada por um ponto;
- `acessa`: operação que devolve as coordenadas de um ponto;
- `atribui`: operação que atribui novos valores às coordenadas de um ponto;
- `distancia`: operação que calcula a distância entre dois pontos.

A interface será dada pelo arquivo `ponto.h`.

```
----
/* TAD: Ponto (x,y) */

/*-----------------*/
/* Tipo exportado */
/*-----------------*/
typedef struct ponto Ponto;

/*--------------------*/
/* Funcoes exportadas */
/*--------------------*/

/* Funcao cria
** Aloca e retorna um ponto com coordenadas (x,y)
```

```
*/
Ponto* cria (float x, float y);

/* Funcao libera
** Libera a memoria de um ponto previamente criado.
*/
void libera (Ponto* p);

/* Funcao acessa
** Devolve os valores das coordenadas de um ponto
*/
void acessa (Ponto* p, float* x, float* y);

/* Funcao atribui
** Atribui novos valores as coordenadas de um ponto
*/
void atribui (Ponto* p, float x, float y);

/* Funcao distancia
** Retorna a distancia entre dois pontos
*/
float distancia (Ponto* p1, Ponto* p2);
```

---

Note que a `struct ponto` não é exportada pelo módulo. Dessa forma, os clientes desse TAD só terão acesso às funções exportadas pelo arquivo `ponto.h`. A implementação do TAD Ponto está no arquivo `ponto.c`.

---

```
#include <stdlib.h>   /* malloc, free, exit */
#include <stdio.h>    /* printf */
#include <math.h>     /* sqrt */
#include "ponto.h"

struct ponto {
  float x;
  float y;
};

Ponto* cria (float x, float y)
{
  Ponto* p = (Ponto*) malloc(sizeof(Ponto));
  if (p == NULL)
  {
    printf("Not enough memory!\n");
    exit(1);
  }
  p->x = x;
  p->y = y;
  return p;
}

void libera (Ponto* p)
{
  free(p);
}

void acessa (Ponto* p, float* x, float* y)
{
  *x = p->x;
```

```
  *y = p->y;
}

void atribui (Ponto* p, float x, float y)
{
  p->x = x;
  p->y = y;
}

float distancia (Ponto* p1, Ponto* p2)
{
  float dx = p2->x - p1->x;
  float dy = p2->y - p1->y;
  return sqrt(dx*dx + dy*dy);
}
----
```

O programa que usa este TAD pode ser o `usaponto.c`.

```
----
#include <stdio.h>
#include "ponto.h"

int main (void)
{
  Ponto *p1, *p2;
  float x1, y1, x2, y2, x, y, d;

  printf("Entre com as coordenadas (x,y) do primeiro ponto: ");
  scanf("%f %f", &x1, &y1);

  printf("Entre com as coordenadas (x,y) do segundo ponto: ");
  scanf("%f %f", &x2, &y2);

  p1 = cria (x1, y1);
  p2 = cria (x2, y2);

  acessa (p1, &x, &y);
  printf("Coordenadas do primeiro ponto: %7.2f %7.2f \n", x, y);

  acessa (p2, &x, &y);
  printf("Coordenadas do segundo ponto : %7.2f %7.2f \n", x, y);

  d = distancia (p1, p2);
  printf("Distancia entre os dois pontos: %7.2f\n", d);
}
----
```

A compilação será:

```
gcc –c ponto.c
gcc –c usaponto.c
gcc –o usaponto usaponto.o ponto.o
```

Note que a implementação do TAD fica completamente escondida.

# EXERCÍCIO

Em todos os TADs abaixo, entregar **o arquivo header** do módulo, o programa C da **implementação** do módulo, e outro programa que **use** esse TAD (ou módulo).

1) Defina um novo TAD Ponto, com as mesmas funções, mas agora representando um ponto em **coordenadas polares**.
2) Defina um novo TAD Ponto, com as mesmas funções, para representar **pontos no $R^3$**.
3) Defina um TAD Matriz (usando memória alocada dinamicamente), com as seguintes operações:
   a) `cria`: operação que cria uma matriz de dimensão m por n;
   b) `libera`: operação que libera a memória alocada para a matriz;
   c) `acessa`: operação que acessa o elemento da linha i e da coluna j da matriz;
   d) `atribui`: operação que atribui o elemento da linha i e da coluna j da matriz;
   e) `linhas`: operação que devolve o número de linhas da matriz;
   f) `colunas`: operação que devolve o número de colunas da matriz.
4) Usando somente as operações definidas na **TAD Matriz**, implemente uma função que dada uma matriz, crie a **matriz transposta** correspondente.

Este trabalho deve ser entregue no dia 06/06/2014. Entretanto, vou liberar a entrega até a data máxima de 09/06/2014 (segunda) até meia-noite.

**IMPORTANTE! Após esta data, o trabalho não será mais aceito**.

O trabalho deve ser possível ser compilado pelo **GCC** e executado no **Linux**.

Envie para o professor (**xbarretox@gmail.com**) e o monitor (**marrco.santos@gmail.com**).

Coloque no assunto: [**LPAV-TP06**].