Universidade Federal do Ceará
Departamento de Computação
Curso de Ciência da Computação

Disciplina	Programação - CK0226	**Semestre**	2022/2
Professor	Lincoln Souza Rocha		
Laboratório de Programação 05 – Tipo Abstratos de Dados			

Descrição Geral do Exercício de Codificação

Comandos para Compilação de Módulos C em Separado:

```
$ gcc -c <tad-impl>.c -o <tad-impl>.o
$ gcc -c <prog-usa-tad>.c -o <prog-usa-tad>.o
$ gcc -o <prog-final>.bin <tad-impl>.o <prog-usa-tad>.o
```

Este exercício consiste em criar um Tipo Abstrato de Dados para manipular figuras geométricas (`geofig`). Para isso, você irá construir tipos estruturados que representam as figuras geométricas e implementar funções que realizam operações sobre essas figuras geométricas. Dessa forma, você deverá exportar os seus tipos estruturados e as suas funções por meio de um arquivo de cabeçalho (`.h`). Os tipos estruturados e as operações (`geofig.h`) a serem codificados (em `geofig.c`) são:

1) Ponto, composto por `x` e `y` do tipo `float`

```
/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;

/* Funções exportadas */

/* Aloca e retorna um ponto com coordenadas (x,y) */
Ponto* pto_cria(float x, float y);

/* Libera a memória de um ponto previamente criado */
void pto_libera(Ponto* p);

/* Copia valores os das coordenadas de um ponto para x e y*/
void pto_acessa Ponto* p, float* x, float* y);

/* Atribui novos valores às coordenadas de um ponto */
void pto_atribui(Ponto* p, float x, float y);

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

2) Círculo, composto por um ponto e um raio do tipo `float`

```
/* TAD: Circulo (ponto,raio) */
/* Tipo exportado */
typedef struct circulo Circulo;

/* Funções exportadas */

/* Aloca e retorna um circulo com base no ponto e no raio informados */
Circulo* circ_cria(Ponto* p, float raio);

/* Libera a memória de um circulo previamente criado */
void circ_libera(Circulo* c);

/* Copia os valores das coordenadas de um circulo e seu raio para x, y e r */
void circ_acessa(Circulo* c, float* x, float* y, float* r);

/* Atribui novos valores às coordenadas de um ponto e seu raio */
void circ_atribui(Circulo* c, float x, float y, float r);
```

```
/* Retorna 1 se o ponto pertence ao circulo ou 0, caso contrário */
int circ_pertence(Circulo* c, Ponto* p);

/* Retorna o cálculo da área do circulo */
float circ_area(Circulo* c);
```

3) Triângulo, composto por três pontos

```
/* TAD: Triangulo (ponto,ponto,ponto) */
/* Tipo exportado */
typedef struct triangulo Triangulo;

/* Funções exportadas */

/* Aloca e retorna um triangulo com base nos pontos */
Triangulo* tria_cria(Ponto* p1, Ponto* p2, Ponto* p3);

/* Libera a memória de um triangulo previamente criado */
void tria_libera(Triangulo* t);

/* Copia os valores dos pontos de um triângulo em p1, p2 e p3*/
void tria_acessa(Triangulo* t, Ponto* p1, Ponto* p2, Ponto* p3);

/* Atribui novos valores aos pontos de um triângulo */
void tria_atribui(Triangulo* t, Ponto* p1, Ponto* p2, Ponto* p3);

/* Retorna 1 se os pontos do triângulo atendem a condição de existência e 0, caso
contrário */
int tria_verifica(Triangulo* t);

/* Retorna 1 se o ponto pertence ao triângulo ou 0, caso contrário */
int tria_pertence (Triangulo* t, Ponto* p);

/* Retorna o cálculo da área do triângulo */
float tria_area(Triangulo* t);
```

4) Quadrilátero, composto por quatro pontos.

```
/* TAD: Quatrilatero (ponto,ponto,ponto,ponto) */
/* Tipo exportado */
typedef struct quatrilatero Quatrilatero;

/* Funções exportadas */

/* Aloca e retorna um quatrilatero com base nos pontos */
Quatrilatero* quad_cria(Ponto* p1, Ponto* p2, Ponto* p3, Ponto* p4);

/* Libera a memória de um quatrilatero previamente criado */
void quad_libera(Quatrilatero* q);

/* Copia os valores dos pontos de um quatrilatero para p1, p2, p3 e p4 */
void quad_acessa(Quatrilatero* q, Ponto* p1, Ponto* p2, Ponto* p3, Ponto* p4);

/* Atribui novos valores aos pontos de um quatrilatero */
void quad_atribui(Quatrilatero* q, Ponto* p1, Ponto* p2, Ponto* p3, Ponto* p4);

/* Retorna 1 se o ponto pertence ao quatrilatero ou 0, caso contrário */
int quad_pertence(Quatrilatero* q, Ponto* p);

/* Retorna o cálculo da área do quatrilatero */
float quad_area(Quatrilatero* q);
```