UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação - ICMC
SCC0202 – Algoritmos e Estrutura de Dados I - 2º Sem /2023
Prof. Rudinei Goularte

Lista de Exercícios Número 1

1. O que é um Tipo Abstrato de Dados (TAD) e qual a característica fundamental na sua utilização?

2. Quais as vantagens de se programar com TADs?

3. Considere dois programas envolvendo um cadastro de funcionários. O programa A foi construído de acordo com os princípios de TAD. Já o programa B não. Diferencie o programa A do programa B.

4. Considere um programa que simule uma calculadora e realize operações de soma, subtração, divisão e multiplicação de números reais. Defina o TAD para este programa. Apresente as condições de entrada e de saída de dados para que as operações possam ser realizadas.

5. Sejam os TADs **Ponto** e **Circulo** como definido abaixo. Desenvolva um programa cliente (main.c) que, pela ordem: crie um ponto **p** e um círculo **r** definidos pelo usuário (stdin); chame a função *distancia(Ponto *p, Circulo *r);* para calcular se **p** é interior ou não a **r**; imprima o resultado. Desenvolva a função *distancia* justificando onde ela deve ser implementada (em que arquivo .c?). Se necessário, desenvolva (implemente) as alterações que julgar pertinentes nos TADs, **sempre justificando**. Um ponto **p** é interior a um círculo **r** se a distância entre **p** e o ponto que define o círculo (ponto_c) é menor que o raio de **r** (raio). A distância entre dois pontos é dada pela equação (1):

$$(1)\, d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

(ponto.h)
```
#ifndef PONTO_H
  #define PONTO_H
  typedef struct ponto_ PONTO;
  PONTO *ponto_criar (float x, float y);
  void ponto_apagar (PONTO *p);
  boolean ponto_set (PONTO *p, float x,
float y);
#endif
```

(ponto.c)
```
...
#include "ponto.h"
struct ponto_{
  float x;
  float y;
};
...
```

(circulo.h)
```
#ifndef CIRCULO_H
 #define CIRCULO_H
 #include "ponto.h"
 typedef struct circulo_ CIRCULO;
 CIRCULO *circulo_criar (float c, float y,
                         float raio);
 void circulo_apagar (CIRCULO *circulo);
 float circulo_area (CIRCULO *circulo);
#endif
```

(circulo.c)
```
...
#include "circulo.h"
#define PI 3.14159
struct circulo_{
   PONTO *ponto_c;
   float raio;
};
...
```

6. O exercício consiste na implementação em C das operações do TAD de números complexos definidas abaixo:

```
/* COMPLEXO_H */
# ifndef COMPLEXO_H
   # define COMPLEXO_H
   typedef struct complexo Complexo ;

   Complexo * criar (double real, double imag);
   void liberar ( Complexo *c);
   void copiar ( Complexo *source, Complexo *target);
   Complexo *adicao ( Complexo *a, Complexo *b);
   Complexo *subtracao ( Complexo *a, Complexo *b);
   int e_real ( Complexo *c);
   double real ( Complexo *c);
   double imag ( Complexo *c);
   void atribuir ( Complexo *c, double real, double imag);
   void atribuir_real ( Complexo *c, double real);
   void atribuir_imag ( Complexo *c, double imag);
# endif
```

Essas operações podem ser definidas da seguinte maneira:

a. Operação criar
A operação criar recebe dois argumentos do tipo *double*, sendo eles, respectivamente, a parte real e a parte imaginária do número complexo. A operação deverá retornar um ponteiro para uma nova estrutura alocada dinamicamente.

b. Operação liberar
A operação liberar recebe como argumento um ponteiro para Complexo. O bloco de memória apontado pelo argumento deverá ser liberado.

c. Operação copiar
A operação copiar recebe como argumento dois ponteiros para Complexo. Os valores do primeiro argumento deverão ser copiados para o segundo argumento. Considera-se que todos os ponteiros apontam para estruturas previamente alocadas.

d. Operações adição e subtração
As operações adição e subtração recebem como argumentos dois ponteiros para Complexo. Os argumentos são os operandos e o retorno o resultado da operação correspondente. Considera-se que todos os ponteiros apontam para estruturas previamente alocadas. Além disso, considere que os valores dos operandos são válidos para a operação.

obs.: também é possível passar um terceiro ponteiro para essas funções e deixar o seu retorno como *void*. O resultado da operação entre os dois primeiros parâmetros é atribuído ao último.

e. Operação e_real
A operação e_real, deverá retornar 1 (TRUE), se a parte imaginária do argumento é nula, ou 0 (FALSE), caso contrário. Considera-se que todos os ponteiros apontam para estruturas previamente alocadas.

f. Operação real e imag
As operações *real* e *imag* retornarão, respectivamente, os valores da parte real e imaginária do argumento. Considera-se que o argumento aponta para uma estrutura previamente alocada.

g. Operação atribuir, atribuir_real e atribuir_imag
A operação atribuir recebe três argumentos, um ponteiro para Complexo e dois valores do tipo *double*. As partes real e imaginária do primeiro argumento deverão receber os valores dos outros parâmetros, respectivamente. O mesmo vale para as operações atribuir_real e atribuir_imag para

cada uma das partes correspondentes. Considera-se que o argumento aponta para uma estrutura previamente alocada.

Fim da lista 1.