

Lista 2 – Estrutura de Dados 1 – 2023.1

Prof. Ana Luiza Bessa de Paula Barros
Ciência da Computação – UECE

- 1) Quais são as duas partes constituintes necessárias para a definição de um TAD?

Resp: Dados e Operações.

- 2) Toda função que compõe um TAD deve receber necessariamente pelo menos um atributo. Qual é este atributo? Justifique sua resposta.

Resp: Não é necessário que uma função de TAD sempre deva receber um atributo como parâmetro. Algumas operações podem não precisar de nenhum atributo, como funções que retornam algum valor derivado dos dados do TAD ou que verificam alguma propriedade de dados. Por exemplo, um TAD “Pessoa” pode ter uma função “getIdade”, que não recebe nenhum parâmetro, mas retorna a idade de uma pessoa armazenada no objeto do tipo “Pessoa”.

- 3) Crie os TADs indicados abaixo:

a) Jogador de Futebol

- i) Cada jogador possui os campos: Nome, Jogos, Gols e Assistências
- ii) Implemente as operações: “Atribuir”, “Imprimir” e “BomJogador”

b) Time de Futebol

- i) Os times possuem os atributos: Nome, Jogadores, Vitórias, Empates e Derrotas
- ii) Implemente as operações: “Atribuir”, “Imprimir” e “Pontuação” (um time de futebol recebe 3 pontos por vitória e 1 ponto por empate)

Resp:

Futebol.h

- Possui uma **struct jogador** com dados **nome**, **jogos**, **gols** e **assistências** e uma **struct time** com dados **nome**, **vitórias**, **empates** e **derrotas**;
- **Operações da struct jogador**: atribui um jogador, com seus dados através de um ponteiro (*ji); **imprime um jogador**, exibindo a performance do mesmo com os dados; e **bom jogador**, que analisa se o jogador está com um desempenho bom o suficiente(nº de gols acima de 10 ou de assistências acima de 5).
- **Operações da struct time**: atribui um time, com seus dados através de um ponteiro (*ti); **imprime um time**, exibindo a performance do mesmo com os dados; e **pontuação**, que disponibiliza a quantidade de pontos do time com base em suas vitórias e empates (**vitória vale 3 pontos e empate vale 1 ponto**);

```
1  #ifndef FUTEBOL_H
2  #define FUTEBOL_H
3
4  typedef struct jogador{
5      char nome[30];
6      int jogos;
7      int gols;
8      int assists;
9
10 }Jogador;
11
12 typedef struct time{
13     char nome[30];
14     Jogador jogadores_time[11];
15     int vitorias;
16     int empates;
17     int derrotas;
18
19 }Time;
20
21 void atribuir_Jogador(Jogador* j, char nome[], int jogos, int gols, int assists);
22 void imprimir_Jogador(Jogador ji);
23 int bom_Jogador(Jogador ji);
24 void atribuir_Time(Time *ti, char nome_time[], Jogador jogadores[], int vitorias, int empates, int derrotas);
25 void imprimir_Time(Time time);
26 void pontuacao(Time time);
27
28 #endif
29
```

Futebol.c

Aplicação da estrutura das operações do TAD com o uso de bibliotecas como **stdio.h**, **stdlib.h**, **string.h** e a importação do protótipo do TAD no **Futebol.h**.

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #include "Futebol.h"
5
6  void atribuir_Jogador(Jogador *ji, char nome[], int jogos, int gols, int assists){
7      strcpy(ji->nome, nome);
8      ji->jogos = jogos;
9      ji->gols = gols;
10     ji->assists = assists;
11
12 }
13
14 void imprimir_Jogador(Jogador ji){
15     printf("Nome: %s\n", ji.nome);
16     printf("Gols: %d\n", ji.gols);
17     printf("Assistências: %d\n", ji.assists);
18     printf("Jogos: %d\n", ji.jogos);
19     printf("-----\n");
20
21
22 }
23
24 int bom_Jogador(Jogador ji){
25     return(ji.gols > 10 || ji.assists > 5);
26 }
27
28 void atribuir_Time(Time *ti, char nome_time[], Jogador jogadores[], int vitorias, int empates, int derrotas){
29     strcpy(ti->nome, nome_time);
30     ti->vitorias = vitorias;
31     ti->derrotas = derrotas;
32     ti->empates = empates;
33     for(int i = 0; i<11; i++){
34         ti->jogadores_time[i] = jogadores[i];
35     }
36
37 }
38
39 void imprimir_Time(Time ti){
40     printf("\n===JOGADOR DESTAQUE===\n");
41     printf("Time: %s\n", ti.nome);
42     printf("Vitórias: %d\n", ti.vitorias);
43     printf("Empates: %d\n", ti.empates);
44     printf("Derrotas: %d\n", ti.derrotas);
45     printf("-----\n");
46 }
47
48 void pontuacao(Time ti){
49     printf("Pontuação Total do Time: %d", ti.vitorias*3 + ti.empates);
50
51 }
52
```

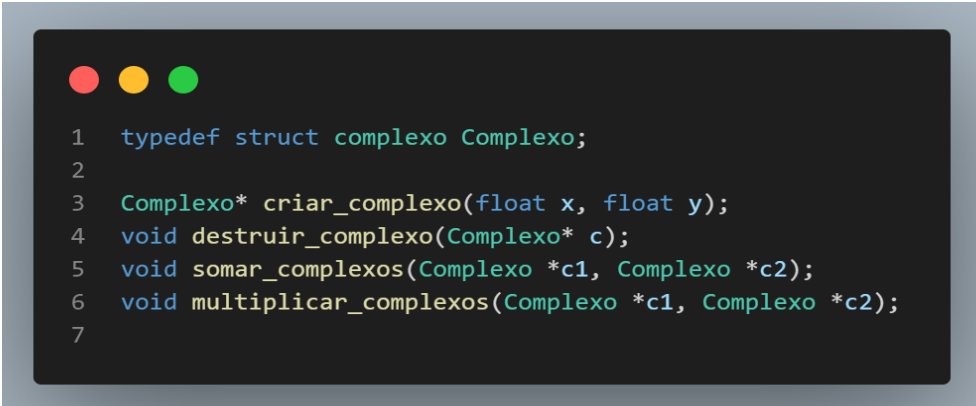
4) Crie um Tipo Abstrato de Dados para representar um **Número Complexo** $z = x + iy$, em que $i = -1$, sendo x a sua parte real e y a parte imaginária.

- Implemente funções para:
 - a) Criar um número complexo
 - b) Destruir um número complexo
 - c) Realizar a soma de dois números complexos
 - d) Realizar a multiplicação de dois números complexos

Resp:

NumCompl.h

- Possui uma **struct complexo** com dados **x(parte real)** e **y(parte imaginária)**;
- **Operações:** **cria um número complexo**, alocando-o na memória com base nos valores de x e y como parâmetros; **destruir um número complexo**, que realiza a operação *free* do mesmo; **somar e multiplicar dois números complexos**, que cria um outro número complexo que é o resultado das operações matemáticas.



```
1 typedef struct complexo Complexo;
2
3 Complexo* criar_complexo(float x, float y);
4 void destruir_complexo(Complexo* c);
5 void somar_complexos(Complexo *c1, Complexo *c2);
6 void multiplicar_complexos(Complexo *c1, Complexo *c2);
7
```

NumComp.c

Aplicação da estrutura das operações do TAD com o uso de bibliotecas como **stdio.h**, **stdlib.h** e a importação do protótipo do TAD no **NumCompl.h**.

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  #include "NumCompl.h"
5
6  struct complexo{
7      float x;
8      float y;
9
10 };
11
12 Complexo* criar_complexo(float real, float imag){
13     Complexo* c = (Complexo*)malloc(sizeof(Complexo));
14     if(c == NULL){
15         printf("Erro na alocação de memória!\n");
16         exit(1);
17     }
18     c->x = real;
19     c->y = imag;
20     return c;
21 }
22
23 void destruir_complexo(Complexo* c){
24     free(c);
25 }
26
27 void somar_complexos(Complexo *c1, Complexo *c2){
28     if(c1 == NULL || c2 == NULL){
29         printf("Um ou os dois números complexos não foram gerados!\n");
30         exit(1);
31     }
32     Complexo soma;
33     soma.x = c1->x + c2->x;
34     soma.y = c1->y + c2->y;
35     if(soma.y < 0){
36         printf("Soma dos Complexos: %.2f + (%.2fi)\n",soma.x,soma.y);
37     }else{
38         printf("Soma dos Complexos: %.2f + %.2fi\n",soma.x,soma.y);
39     }
40 }
41
42 void multiplicar_complexos(Complexo *c1, Complexo *c2){
43     if(c1 == NULL || c2 == NULL){
44         printf("Um ou os dois números complexos não foram gerados!\n");
45         exit(1);
46     }
47     Complexo mult;
48     mult.x = c1->x*c2->x - c1->y*c2->y;
49     mult.y = c1->x*c2->y + c1->y*c2->x;
50     if(mult.y < 0){
51         printf("Produto dos Complexos: %.2f + (%.2fi)\n",mult.x,mult.y);
52     }else{
53         printf("Produto dos Complexos: %.2f + %.2fi\n",mult.x,mult.y);
54     }
55 }
56
57 }
58
59 }

```

- 5) Crie um Tipo Abstrato de Dados para representar uma **Esfera**. Inclua as funções de inicialização necessárias e as operações que retornem o seu raio, a sua área e o seu volume.



```
1  typedef struct esfera Esfera;
2
3  Esfera* cria_Esfera(float r);
4  void libera_Esfera(Esfera *e);
5  void raio_Esfera(Esfera *e);
6  void volume_Esfera(Esfera *e);
```



```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "Esfera.h"
4
5  struct esfera{
6      float raio;
7
8  };
9
10 Esfera* cria_Esfera(float r){
11     Esfera* e = (Esfera*)malloc(sizeof(Esfera));
12     if(e != NULL){
13         e->raio = r;
14     }
15     return e;
16 }
17
18 void libera_Esfera(Esfera *e){
19     free(e);
20 }
21
22 void raio_Esfera(Esfera *e){
23     if(e == NULL){
24         printf("Erro: a esfera não foi criada!\n");
25         return;
26     }
27     printf("Raio da Esfera: %.2f cm\n",e->raio);
28 }
29
30 void volume_Esfera(Esfera *e){
31     if(e == NULL){
32         printf("Erro: a esfera não foi criada!\n");
33         return;
34     }
35     printf("Volume da Esfera: %.2f cm^3\n", (4*(3.14159)*(e->raio*e->raio*e->raio))/3);
36 }
37
38 }
```

6) Dado o TAD **Ponto** descrito abaixo:

```
/** Libera a memória de um ponto previamente criado.
 */
void libera (Ponto* p);
/* Função acessa
 ** Devolve os valores das coordenadas de um ponto
 */
void acessa (Ponto* p, float* x, float* y);
/* Função atribui
 ** Atribui novos valores às coordenadas de um ponto
 */
void atribui (Ponto* p, float x, float y);
/* Função distancia
 ** Retorna a distância entre dois pontos
 */
float distancia (Ponto* p1, Ponto* p2);
```

```
#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("Memória insuficiente!\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);
}
```

- a) Escreva um programa que faça uso do TAD **Ponto**
- b) Acrescente novas operações ao TAD **Ponto**, tais como soma e subtração de pontos

Rep:

- a) **Implementação do TAD Ponto já com as funções Soma e Subtração inseridas.** Importa-se **stdio.h** (biblioteca padrão), **stdlib.h** (alocação de memória) e **"Ponto.c"** (para o uso da estrutura das funções implementadas e com protótipo no **"Ponto.h"**).

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "Ponto.c"
4
5  int main(){
6      float d;
7      Ponto *p, *q;
8      float x1, x2, y1, y2;
9      printf("Insira os parâmetros do primeiro ponto: ");
10     scanf("%f",&x1);
11     scanf("%f",&y1);
12     printf("Insira os parâmetros do segundo ponto: ");
13     scanf("%f",&x2);
14     scanf("%f",&y2);
15     p = Ponto_cria(x1, y1);
16     q = Ponto_cria(x2, y2);
17     d = Ponto_distancia(p,q);
18     printf("Distância entre os pontos: %.2f\n", d);
19     Ponto_soma(p,q);
20     Ponto_subtrai(p,q);
21     Ponto_libera(q);
22     Ponto_libera(p);
23     return 0;
24 }
25
26
```

- b) **Implementação das funções Soma e Subtração.** Resumem-se na seguinte descrição : um **if** que analise se os pontos foram alocados na memória. Se não, o programa se encerra e libera a mensagem de erro. Caso contrário, um novo **Ponto soma** ou **Ponto sub** será criado, com suas coordenadas **x** e **y** recebendo as respectivas soma e subtração **das coordenadas dos dois pontos parâmetros**. Depois o resultado é imprimido.



```
1 void Ponto_soma(Ponto *p1, Ponto* p2){
2     if(p1 == NULL || p2 == NULL){
3         printf("Erro: um ou os dois pontos não foram criados!\n");
4         return;
5     }
6     Ponto soma;
7     soma.x = p1->x + p2->x;
8     soma.y = p1->y + p2->y;
9     printf("\n==SOMA DE PONTOS==\n");
10    printf("Coordenada x: %.2f\n", soma.x);
11    printf("Coordenada y: %.2f\n\n", soma.y);
12 }
13
14 void Ponto_subtrai(Ponto *p1, Ponto* p2){
15     if(p1 == NULL || p2 == NULL){
16         printf("Erro: um ou os dois pontos não foram criados!\n");
17         return;
18     }
19     Ponto sub;
20     sub.x = p2->x - p1->x;
21     sub.y = p2->y - p1->y;
22     printf("\n==SUBTRAÇÃO DE PONTOS==\n");
23     printf("Coordenada x: %.2f\n", sub.x);
24     printf("Coordenada y: %.2f\n\n", sub.y);
25 }
26
```