

# UFSJ - Ciências da Computação

## Laboratório de Programação 2

### Roteiro 3

**Nome: Geraldo Arthur Detomi**

#### Exercício 1.1

Um tipo abstrato de dados, ou TAD, é um conjunto de dados estruturados e as operações que podem ser executadas sobre esses dados. Tanto a representação quanto as operações do TAD são especificadas pelo programador. O usuário utiliza o TAD como uma caixa-preta, por meio de sua interface. As vantagens da utilização do TAD são: encapsulamento (ao ocultar a implementação, fornece um conjunto de operações possíveis para o TAD, e isso é tudo que o usuário precisa saber), segurança (o usuário não tem acesso aos dados, o que evita que ele os manipule de forma incorreta), flexibilidade (pode-se alterar o TAD sem alterar as aplicações que o utilizam, sendo possível ter diferentes implementações de um TAD, desde que todas respeitem a interface) e reutilização (a implementação do TAD é feita em um módulo diferente do programa do usuário).

#### Exercício 1.2

**roteiro\_3/cubo.h**

```
1  #ifndef CUBO_H
2  #define CUBO_H
3
4  typedef struct cubo Cubo;
5
6  Cubo *criar_cubo(double lado);
7
8  double get_lado_cubo(Cubo *cubo);
9
10 double get_area_cubo(Cubo *cubo);
11
12 double get_volume_cubo(Cubo *cubo);
13
14 void liberar_cubo(Cubo *cubo);
15
16 #endif
```

**roteiro\_3/cubo.c**

```
1  #include "cubo.h"
2  #include <math.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  struct cubo {
7      double lado;
8  };
```

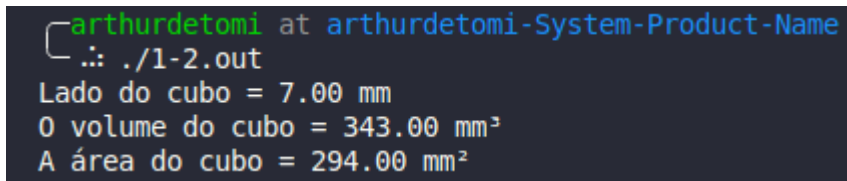
```
9
10 void verificar_estouro_memoria(void *ponteiro) {
11     if (ponteiro == NULL) {
12         printf("Erro presença de ponteiro nulo em local indevido\n");
13         exit(1);
14     }
15 }
16
17 Cubo *criar_cubo(double lado) {
18     Cubo *cubo = (Cubo *)malloc(sizeof(Cubo));
19
20     verificar_estouro_memoria(cubo);
21
22     cubo->lado = lado;
23
24     return cubo;
25 }
26
27 double get_lado_cubo(Cubo *cubo) {
28     verificar_estouro_memoria(cubo);
29
30     return cubo->lado;
31 }
32
33 double get_volume_cubo(Cubo *cubo) {
34     verificar_estouro_memoria(cubo);
35
36     return pow(cubo->lado, 3);
37 }
38
39 double get_area_cubo(Cubo *cubo) {
40     verificar_estouro_memoria(cubo);
41
42     double area_base = pow(cubo->lado, 2);
43
44     double area_lateral = 4 * area_base;
45
46     return 2 * area_base + area_lateral;
47 }
48
49 void liberar_cubo(Cubo *cubo) {
50     free(cubo);
51     cubo = NULL;
52 }
```

### roteiro\_3/1-2.c

```
1 #include "cubo.h"
2 #include <stdio.h>
3
4 int main() {
5     Cubo *cubo = criar_cubo(7);
6
7     printf("Lado do cubo = %.2lf mm\n", get_lado_cubo(cubo));
```

```
8   printf("O volume do cubo = %.2lf mm³\n", get_volume_cubo(cubo));
9   printf("A área do cubo = %.2lf mm²\n", get_area_cubo(cubo));
10
11   liberar_cubo(cubo);
12
13   return 0;
14 }
```

Saída do terminal:



```
arthurdetomi at arthurdetomi-System-Product-Name
.: ./1-2.out
Lado do cubo = 7.00 mm
O volume do cubo = 343.00 mm³
A área do cubo = 294.00 mm²
```

## Exercício 1.3

roteiro\_3/conjunto.h

```
1  #ifndef CONJUNTO_H
2  #define CONJUNTO_H
3
4  #include <stdbool.h>
5
6  typedef int elemento;
7
8  typedef struct conjunto Conjunto;
9
10 Conjunto *criar_conjunto();
11
12 Conjunto *unir_conjuntos(Conjunto *c1, Conjunto *c2);
13
14 void inserir(Conjunto *c1, elemento elemento);
15
16 void remover(Conjunto *c1, elemento elemento);
17
18 Conjunto *get_intersecao_conjuntos(Conjunto *c1, Conjunto *c2);
19
20 Conjunto *get_diferenca_conjuntos(Conjunto *c1, Conjunto *c2);
21
22 bool is_pertence_ao_conjunto(Conjunto *c, elemento elemento);
23
24 elemento get_menor_valor_conjunto(Conjunto *c);
25
26 elemento get_maior_valor_conjunto(Conjunto *c);
27
28 bool is_equal(Conjunto *c1, Conjunto *c2);
29
30 int get_tamanho_conjunto(Conjunto *c);
31
32 bool is_empty_conjunto(Conjunto *c);
33
34 void liberar_conjunto(Conjunto *c);
35
```

```
36 void imprimir_conjunto(Conjunto *c);
37
38 #endif
```

### roteiro\_3/conjunto.c

```
1  #include "conjunto.h"
2  #include <stdbool.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define MAX 100
7
8  struct conjunto {
9      elemento *elementos;
10     int tamanho;
11     int max_size;
12 };
13
14 void verificar_estouro_memoria(void *ponteiro) {
15     if (ponteiro == NULL) {
16         printf("Erro presença de ponteiro nulo em local indevido\n");
17         exit(1);
18     }
19 }
20
21 Conjunto *criar_conjunto() {
22     Conjunto *conjunto = (Conjunto *)malloc(sizeof(Conjunto));
23
24     verificar_estouro_memoria(conjunto);
25
26     conjunto->tamanho = 0;
27     conjunto->max_size = MAX;
28
29     conjunto->elementos =
30         (elemento *)malloc(conjunto->max_size * sizeof(elemento));
31
32     verificar_estouro_memoria(conjunto->elementos);
33
34     return conjunto;
35 }
36
37 Conjunto *unir_conjuntos(Conjunto *c1, Conjunto *c2) {
38     Conjunto *novo_conjunto = criar_conjunto();
39
40     for (int i = 0; i < c1->tamanho; i++) {
41         inserir(novo_conjunto, c1->elementos[i]);
42     }
43
44     for (int i = 0; i < c2->tamanho; i++) {
45         inserir(novo_conjunto, c2->elementos[i]);
46     }
47
48     return novo_conjunto;
```

```
49 }
50
51 // Inserir elementos no conjunto não permitindo duplicatas
52 void inserir(Conjunto *c1, elemento el) {
53     verificar_estouro_memoria(c1);
54
55     if (is_pertence_ao_conjunto(c1, el)) {
56         return;
57     }
58
59     if (c1->tamanho + 1 >= c1->max_size) {
60         c1->elementos =
61             (elemento *)realloc(c1->elementos, c1->max_size * 2 * sizeof(elemento));
62     }
63
64     c1->elementos[c1->tamanho] = el;
65
66     c1->tamanho++;
67 }
68
69 void remover(Conjunto *c1, elemento elemento) {
70     verificar_estouro_memoria(c1);
71
72     int index = -1;
73
74     for (int i = 0; i < c1->tamanho; i++) {
75         if (elemento == c1->elementos[i]) {
76             index = i;
77             break;
78         }
79     }
80
81     if (index == -1) {
82         return;
83     }
84
85     for (int i = index; i < c1->tamanho - 1; i++) {
86         c1->elementos[i] = c1->elementos[i + 1];
87     }
88
89     c1->tamanho--;
90 }
91
92 Conjunto *get_intersecao_conjuntos(Conjunto *c1, Conjunto *c2) {
93     verificar_estouro_memoria(c1);
94     verificar_estouro_memoria(c2);
95
96     Conjunto *novo_conjunto = criar_conjunto();
97
98     Conjunto *menor_conjunto = (c1->tamanho < c2->tamanho) ? c1 : c2;
99     Conjunto *maior_conjunto = (c2->tamanho > c1->tamanho) ? c2 : c1;
100
101     for (int i = 0; i < menor_conjunto->tamanho; i++) {
102         elemento elemento_atual = menor_conjunto->elementos[i];
```

```
103
104     if (is_pertence_ao_conjunto(maior_conjunto, elemento_atual)) {
105         inserir(novo_conjunto, elemento_atual);
106     }
107 }
108
109 return novo_conjunto;
110 }
111
112 Conjunto *get_diferenca_conjuntos(Conjunto *c1, Conjunto *c2) {
113     verificar_estouro_memoria(c1);
114     verificar_estouro_memoria(c2);
115
116     Conjunto *novo_conjunto = criar_conjunto();
117
118     Conjunto *menor_conjunto = (c1->tamanho < c2->tamanho) ? c1 : c2;
119     Conjunto *maior_conjunto = (c2->tamanho > c1->tamanho) ? c2 : c1;
120
121     for (int i = 0; i < maior_conjunto->tamanho; i++) {
122         elemento elemento_atual = maior_conjunto->elementos[i];
123
124         if (!is_pertence_ao_conjunto(menor_conjunto, elemento_atual)) {
125             inserir(novo_conjunto, elemento_atual);
126         }
127     }
128
129     return novo_conjunto;
130 }
131
132 bool is_pertence_ao_conjunto(Conjunto *c, elemento el) {
133     verificar_estouro_memoria(c);
134     for (int i = 0; i < c->tamanho; i++) {
135         if (c->elementos[i] == el) {
136             return true;
137         }
138     }
139     return false;
140 }
141
142 elemento get_menor_valor_conjunto(Conjunto *c) {
143     verificar_estouro_memoria(c);
144
145     elemento menor = c->elementos[0];
146
147     for (int i = 1; i < c->tamanho; i++) {
148         if (c->elementos[i] < menor) {
149             menor = c->elementos[i];
150         }
151     }
152
153     return menor;
154 }
155
156 elemento get_maior_valor_conjunto(Conjunto *c) {
```

```
157     verificar_estouro_memoria(c);
158
159     elemento maior = c->elementos[0];
160
161     for (int i = 1; i < c->tamanho; i++) {
162         if (c->elementos[i] > maior) {
163             maior = c->elementos[i];
164         }
165     }
166
167     return maior;
168 }
169
170 bool is_igual(Conjunto *c1, Conjunto *c2) {
171     verificar_estouro_memoria(c1);
172     verificar_estouro_memoria(c2);
173
174     if (c1->tamanho != c2->tamanho) {
175         return false;
176     }
177
178     for (int i = 0; i < c1->tamanho; i++) {
179         if (!is_pertence_ao_conjunto(c2, c1->elementos[i])) {
180             return false;
181         }
182     }
183
184     return true;
185 }
186
187 int get_tamanho_conjunto(Conjunto *c) {
188     verificar_estouro_memoria(c);
189
190     return c->tamanho;
191 }
192
193 bool is_empty_conjunto(Conjunto *c) {
194     verificar_estouro_memoria(c);
195     return c->tamanho == 0;
196 }
197
198 void liberar_conjunto(Conjunto *c) {
199     free(c->elementos);
200     c->elementos = NULL;
201
202     free(c);
203     c = NULL;
204 }
205
206 void imprimir_conjunto(Conjunto *c) {
207     printf("[ ");
208     for (int i = 0; i < c->tamanho; i++) {
209         printf("%d ", c->elementos[i]);
210         if (i != c->tamanho - 1) {
```

```
211     printf(",");
212 }
213 }
214 printf("]\n");
215 }
```

### roteiro\_3/1-3.c

```
1  #include "conjunto.h"
2  #include <stdbool.h>
3  #include <stdio.h>
4
5  #define MAX 5
6
7  int main() {
8      enum opcoes {
9          UNIAO = 0,
10         INTERSECAO = 1,
11         DIFERENCA = 2,
12         IGUAIS = 3,
13         SAIR = -1
14     };
15
16     printf("Testes de funções básicas:\n");
17     Conjunto *conjunto_teste = criar_conjunto();
18
19     for (int i = 1; i <= 5; i++) {
20         inserir(conjunto_teste, i);
21     }
22
23     printf("Conjunto de teste:\n");
24     imprimir_conjunto(conjunto_teste);
25
26     printf("Maior valor = %d\n", get_maior_valor_conjunto(conjunto_teste));
27     printf("Menor valor = %d\n", get_menor_valor_conjunto(conjunto_teste));
28
29     remover(conjunto_teste, 1);
30     printf("Conjunto após remover o número 1\n");
31     imprimir_conjunto(conjunto_teste);
32
33     liberar_conjunto(conjunto_teste);
34
35     Conjunto *c1 = criar_conjunto(), *c2 = criar_conjunto();
36
37     printf("Insira %d elementos no conjunto 1:\n", MAX);
38     for (int i = 0; i < MAX; i++) {
39         int value;
40         scanf("%d", &value);
41
42         inserir(c1, value);
43     }
44
45     printf("Insira %d elementos no conjunto 2:\n", MAX);
46     for (int i = 0; i < MAX; i++) {
```



```
47     int value;
48     scanf("%d", &value);
49
50     inserir(c2, value);
51 }
52
53 while (1) {
54     printf("Digite a operação:\n");
55
56     printf("UNIAO[%d]\n", UNIAO);
57     printf("INTERSECAO[%d]\n", INTERSECAO);
58     printf("DIFERENCA[%d]\n", DIFERENCA);
59     printf("IGUAIS[%d]\n", IGUAIS);
60     printf("SAIR[%d]\n", SAIR);
61
62     int input;
63
64     printf("Opção : ");
65     scanf("%d", &input);
66
67     Conjunto *resultado;
68
69     switch (input) {
70     case UNIAO:
71         printf("UNIAO[%d] choosed:\n", UNIAO);
72         resultado = unir_conjuntos(c1, c2);
73         printf("veio aqui\n");
74         imprimir_conjunto(resultado);
75         liberar_conjunto(resultado);
76
77         break;
78     case INTERSECAO:
79         printf("INTERSECAO[%d] choosed:\n", INTERSECAO);
80         resultado = get_intersecao_conjuntos(c1, c2);
81         imprimir_conjunto(resultado);
82         liberar_conjunto(resultado);
83
84         break;
85     case DIFERENCA:
86         printf("DIFERENCA[%d] choosed:\n", DIFERENCA);
87         resultado = get_diferenca_conjuntos(c1, c2);
88         imprimir_conjunto(resultado);
89         liberar_conjunto(resultado);
90
91         break;
92     case IGUAIS:
93         printf("IGUAIS[%d] choosed:\n", IGUAIS);
94         bool iguais = is_equal(c1, c2);
95
96         if (iguais) {
97             printf("Os conjuntos são iguais\n");
98         } else {
99             printf("Os conjuntos não são iguais\n");
100         }
101     }
```

```
101  
102     break;  
103     case SAIR:  
104         return 0;  
105     }  
106  
107     printf("\n\n");  
108 }  
109  
110 liberar_conjunto(c1);  
111 liberar_conjunto(c2);  
112  
113 return 0;  
114 }
```

**Saída do terminal:**

```
arthurdetomi at arthurdetomi-System-  
└─$ ./1-3.out  
Testes de funções básicas:  
Conjunto de teste:  
[ 1 ,2 ,3 ,4 ,5 ]  
Maior valor = 5  
Menor valor = 1  
Conjunto após remover o número 1  
[ 2 ,3 ,4 ,5 ]  
Insira 5 elementos no conjunto 1:  
1  
2  
3  
4  
5  
Insira 5 elementos no conjunto 2:  
1  
2  
3  
8  
9  
Digite a operação:  
UNIAO[0]  
INTERSECAO[1]  
DIFERENCA[2]  
IGUAIS[3]  
SAIR[-1]  
Opção : 0  
UNIAO[0] choosed:  
veio aqui  
[ 1 ,2 ,3 ,4 ,5 ,8 ,9 ]  
  
Digite a operação:  
UNIAO[0]  
INTERSECAO[1]  
DIFERENCA[2]  
IGUAIS[3]  
SAIR[-1]  
Opção : 1  
INTERSECAO[1] choosed:  
[ 1 ,2 ,3 ]  
  
Digite a operação:  
UNIAO[0]  
INTERSECAO[1]  
DIFERENCA[2]  
IGUAIS[3]  
SAIR[-1]  
Opção : 2  
DIFERENCA[2] choosed:  
[ 4 ,5 ]  
  
Digite a operação:  
UNIAO[0]  
INTERSECAO[1]  
DIFERENCA[2]  
IGUAIS[3]  
SAIR[-1]  
Opção : 3  
IGUAIS[3] choosed:  
Os conjuntos não são iguais
```