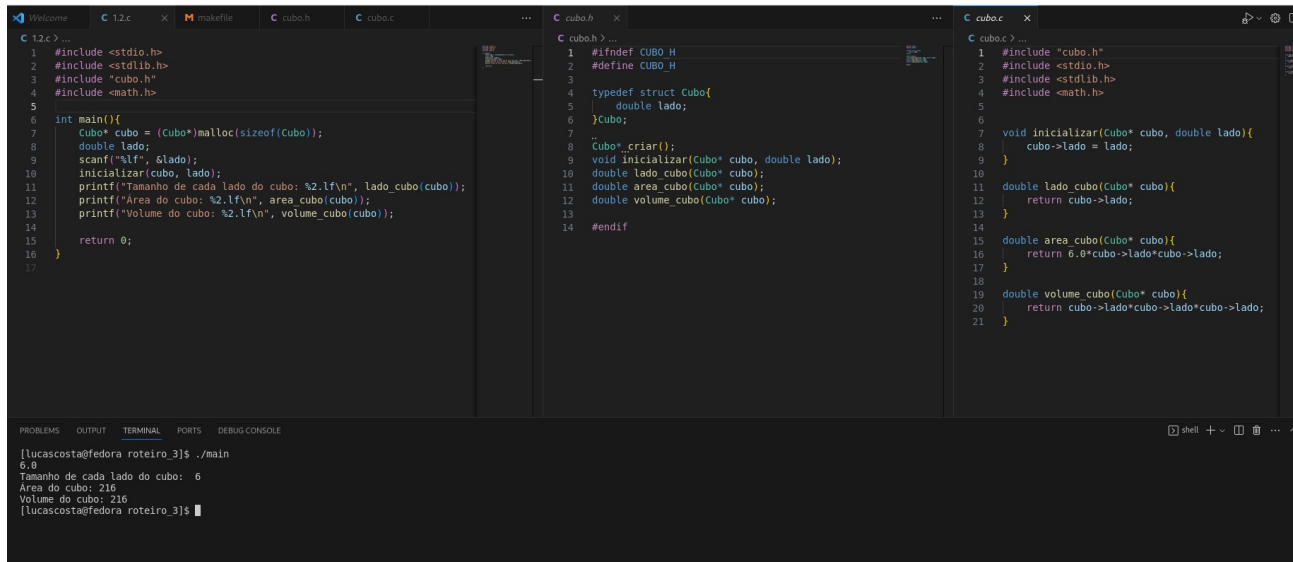


ROTEIRO 3 – LUCAS EDUARDO LEITE COSTA

1.1 Os Tipos Abstratos de Dados (TAD) é um tipo de dados definido pelo programador, que tem como intuito definir um conjunto de valores e operações relacionadas à esses dados. Com isso, as principais vantagens da utilização de TAD's são relacionadas ao encapsulamento das funções e a abstração de informações.

1.2



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "cubo.h"
4 #include <math.h>
5
6 int main(){
7     Cubo* cubo = (Cubo*)malloc(sizeof(Cubo));
8     double lado;
9     scanf("%lf", &lado);
10    inicializar(cubo, lado);
11    printf("Tamanho de cada lado do cubo: %2.1f\n", lado_cubo(cubo));
12    printf("Área do cubo: %2.1f\n", area_cubo(cubo));
13    printf("Volume do cubo: %2.1f\n", volume_cubo(cubo));
14
15    return 0;
16 }
17
```

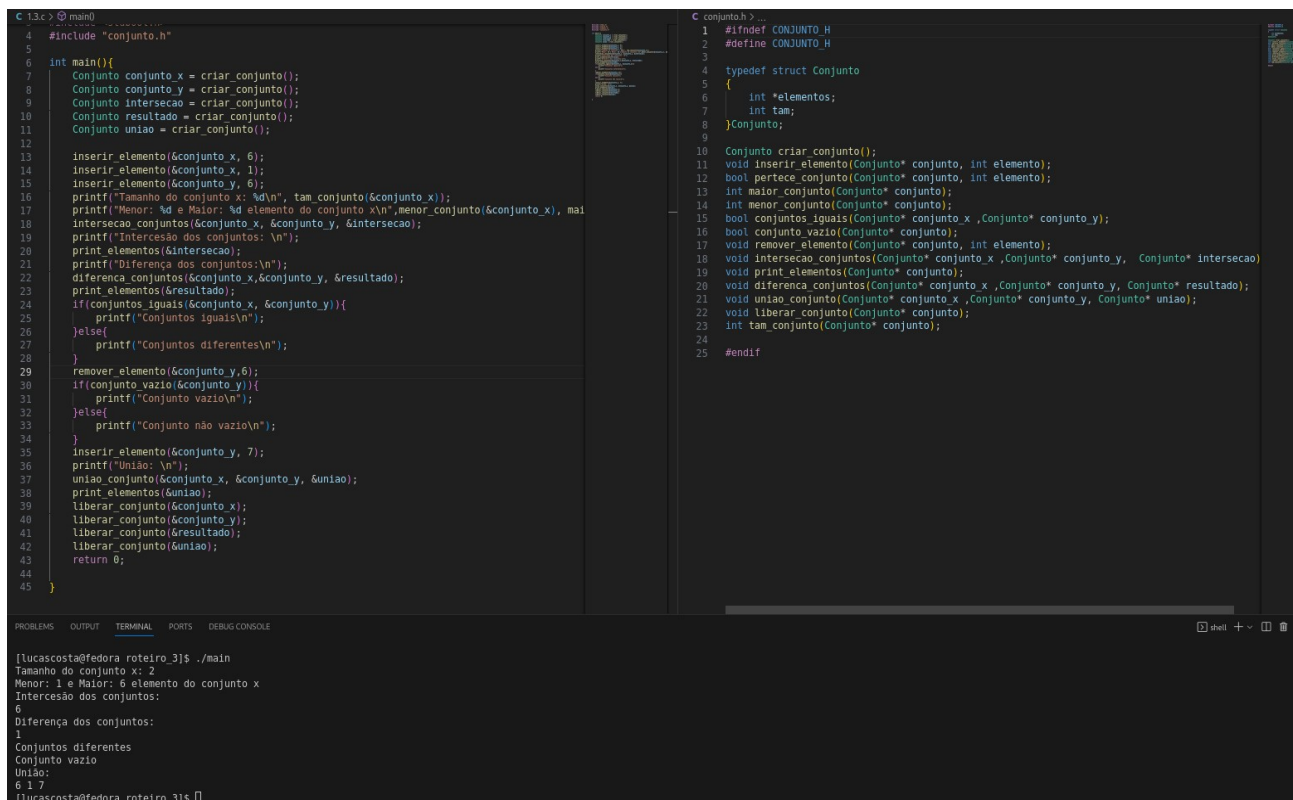
```
1 #ifndef CUBO_H
2 #define CUBO_H
3
4 typedef struct Cubo{
5     double lado;
6 }Cubo;
7
8 Cubo* criar();
9 void inicializar(Cubo* cubo, double lado);
10 double lado_cubo(Cubo* cubo);
11 double area_cubo(Cubo* cubo);
12 double volume_cubo(Cubo* cubo);
13
14 #endif

```

```
1 #include "cubo.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5
6 void inicializar(Cubo* cubo, double lado){
7     cubo->lado = lado;
8 }
9
10 double lado_cubo(Cubo* cubo){
11     return cubo->lado;
12 }
13
14 double area_cubo(Cubo* cubo){
15     return 6.0*cubo->lado*cubo->lado;
16 }
17
18 double volume_cubo(Cubo* cubo){
19     return cubo->lado*cubo->lado*cubo->lado;
20 }
21
```

```
[lucas@fedora roteiro_3]$ ./main
6.0
Tamanho de cada lado do cubo: 6
Área do cubo: 216
Volume do cubo: 216
[lucas@fedora roteiro_3]$
```

1.3



```
1 #include "conjunto.h"
2
3 int main(){
4     Conjunto conjunto_x = criar_conjunto();
5     Conjunto conjunto_y = criar_conjunto();
6     Conjunto intersecao = criar_conjunto();
7     Conjunto resultado = criar_conjunto();
8     Conjunto uniao = criar_conjunto();
9
10    inserir_elemento(&conjunto_x, 6);
11    inserir_elemento(&conjunto_x, 1);
12    inserir_elemento(&conjunto_y, 6);
13    printf("Tamanho do conjunto x: %d\n", tam_conjunto(&conjunto_x));
14    printf("Menor: %d e Maior: %d elemento do conjunto x\n", menor_conjunto(&conjunto_x), maior_conjunto(&conjunto_x));
15    intersecao_conjuntos(&conjunto_x, &conjunto_y, &intersecao);
16    printf("Interseção dos conjuntos: \n");
17    print_elementos(&intersecao);
18    printf("Diferença dos conjuntos:\n");
19    diferenca_conjuntos(&conjunto_x, &conjunto_y, &resultado);
20    print_elementos(&resultado);
21    if(conjuntos_iguais(&conjunto_x, &conjunto_y)){
22        printf("Conjuntos iguais\n");
23    }
24    else{
25        printf("Conjuntos diferentes\n");
26    }
27    remover_elemento(&conjunto_y, 6);
28    if(conjunto_vazio(&conjunto_y)){
29        printf("Conjunto vazio\n");
30    }
31    else{
32        printf("Conjunto não vazio\n");
33    }
34    inserir_elemento(&conjunto_y, 7);
35    printf("União: \n");
36    uniao_conjuntos(&conjunto_x, &conjunto_y, &uniao);
37    print_elementos(&uniao);
38    liberar_conjunto(&conjunto_x);
39    liberar_conjunto(&conjunto_y);
40    liberar_conjunto(&resultado);
41    liberar_conjunto(&uniao);
42
43    return 0;
44 }
45
```

```
1 #ifndef CONJUNTO_H
2 #define CONJUNTO_H
3
4 typedef struct Conjunto
5 {
6     int *elementos;
7     int tam;
8 }Conjunto;
9
10 Conjunto criar_conjunto();
11 void inserir_elemento(Conjunto* conjunto, int elemento);
12 bool pertence_conjunto(Conjunto* conjunto, int elemento);
13 int maior_conjunto(Conjunto* conjunto);
14 int menor_conjunto(Conjunto* conjunto);
15 bool conjuntos_iguais(Conjunto* conjunto_x, Conjunto* conjunto_y);
16 bool conjunto_vazio(Conjunto* conjunto);
17 void remover_elemento(Conjunto* conjunto, int elemento);
18 void intersecao_conjuntos(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* intersecao);
19 void print_elementos(Conjunto* conjunto);
20 void diferenca_conjuntos(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* resultado);
21 void uniao_conjuntos(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* uniao);
22 void liberar_conjunto(Conjunto* conjunto);
23 int tam_conjunto(Conjunto* conjunto);
24
25 #endif

```

```
[lucas@fedora roteiro_3]$ ./main
Tamanho do conjunto x: 2
Menor: 1 e Maior: 6 elemento do conjunto x
Interseção dos conjuntos:
6
Diferença dos conjuntos:
1
Conjuntos diferentes
Conjunto vazio
União:
6 1 7
[lucas@fedora roteiro_3]$
```

```

C conjunto.c > @ intersecao_conjuntos(Conjunto *, Conjunto *, Conjunto *)
3 #include <stdbool.h>
4 #include "conjunto.h"
5
6 Conjunto criar_conjunto(){
7     Conjunto conjunto;
8     conjunto.elementos = NULL;
9     conjunto.tam = 0;
10    return conjunto;
11}
12
13 void inserir_elemento(Conjunto* conjunto, int elemento){
14     for(int i = 0; i < conjunto->tam; i++){
15         if(conjunto->elementos[i] == elemento)
16             return;
17     }
18     conjunto->tam++;
19     conjunto->elementos = (int*)realloc(conjunto->elementos, conjunto->tam*sizeof(int));
20     conjunto->elementos[conjunto->tam-1] = elemento;
21}
22
23 bool pertence_conjunto(Conjunto* conjunto, int elemento){
24     for(int i = 0; i < conjunto->tam; i++){
25         if(conjunto->elementos[i] == elemento)
26             return true;
27     }
28     return false;
29}
30
31 int maior_conjunto(Conjunto* conjunto){
32     int maior = conjunto->elementos[0];
33     for(int i = 0; i < conjunto->tam; i++){
34         if(conjunto->elementos[i] > maior)
35             maior = conjunto->elementos[i];
36     }
37     return maior;
38}
39
40 int menor_conjunto(Conjunto* conjunto){
41     int menor = conjunto->elementos[0];
42     for(int i = 0; i < conjunto->tam; i++){
43         if(conjunto->elementos[i] < menor)
44             menor = conjunto->elementos[i];
45     }
46     return menor;
47}
48
49 bool conjuntos_iguais(Conjunto* conjunto_x, Conjunto* conjunto_y){
50     if(conjunto_x->tam != conjunto_y->tam){
51         return false;
52     }
53     for(int i = 0; i < conjunto_x->tam; i++){
54         if(false == pertence_conjunto(conjunto_y, conjunto_x->elementos[i]))
55             return false;
56     }
57     return true;
58}
59
60 bool conjunto_vazio(Conjunto* conjunto){
61     if(conjunto->tam == 0)
62         return true;
63     return false;
64}
65
66 void remover_elemento(Conjunto* conjunto, int elemento){
67     for(int i = 0; i < conjunto->tam; i++){
68         if(conjunto->elementos[i] == elemento)
69             for(int j = 0; j < conjunto->tam - i; j++){
70                 conjunto->elementos[j] = conjunto->elementos[j+1];
71             }
72         conjunto->elementos = (int*)realloc(conjunto->elementos, conjunto->tam*sizeof(int));
73         conjunto->tam--;
74     }
75     return;
76}
77
78 int tam_conjunto(Conjunto* conjunto){
79     return conjunto->tam;
80}
81
82 void intersecao_conjuntos(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* intersecao){
83     for(int i = 0; i < conjunto_x->tam; i++){
84         int elemento = conjunto_x->elementos[i];
85         if(pertence_conjunto(conjunto_y, elemento))
86             inserir_elemento(intersecao, elemento);
87     }
88}
89
90 void print_elementos(Conjunto* conjunto){
91     for(int i = 0; i < conjunto->tam; i++){
92

```

```

C conjunto.c > @ intersecao_conjuntos(Conjunto *, Conjunto *, Conjunto *)
87 void print_elementos(Conjunto* conjunto){
88     for(int i=0; i<conjunto->tam; i++){
89         printf("%d ", conjunto->elementos[i]);
90     }
91     printf("\n");
92 }
93
94 void diferenca_conjuntos(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* resultado){
95     for(int i=0; i<conjunto_x->tam; i++){
96         int elemento = conjunto_x->elementos[i];
97         if(false == pertence_conjunto(conjunto_y, elemento))
98             inserir_elemento(resultado, elemento);
99     }
100 }
101
102 void uniao_conjunto(Conjunto* conjunto_x, Conjunto* conjunto_y, Conjunto* uniao){
103     for(int i=0; i<conjunto_x->tam; i++){
104         inserir_elemento(uniao, conjunto_x->elementos[i]);
105     }
106
107     for(int i=0; i<conjunto_y->tam; i++){
108         if(false==pertence_conjunto(uniao,conjunto_y->elementos[i]))
109             inserir_elemento(uniao, conjunto_y->elementos[i]);
110     }
111 }
112
113 void liberar_conjunto(Conjunto* conjunto){
114     free(conjunto->elementos);
115     conjunto->tam=0;
116     conjunto->elementos=NULL;
117 }

```