

① O conceito TAD (tipo Abstrato de Dados) é um conjunto de valores com seu comportamento definido por operações implementadas na forma de funções. Na linguagem C é feito criando um arquivo .h e um .c, onde no .h fica o cabeçalho das funções e estruturas e no .c contém a implementação das funções que estão declaradas no arquivo h.

• Vantagens:

- Encapsulamento: o encapsulamento ajuda a reduzir a complexidade interna e protege os dados contra acesso não autorizado e também é bom para reutiliza funções e estruturas em outros arquivos.
- Manutenção: conseguimos alterar o TDA sem alterar as aplicações que o utilizam.

O TAD é realizado em arquivos diferentes, pois fica fácil para a manutenção, organização o código fica mais limpo e fácil de entender, Abstração escondendo os dados o máximo possível, e reutilização do código.

② F0: 5, 7, 12, 4, 0, 4, 6, 8, 67, 34, 23, 5, 0, 44, 33, 22, 6, 0

Após o código:

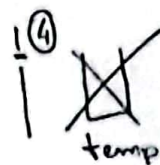
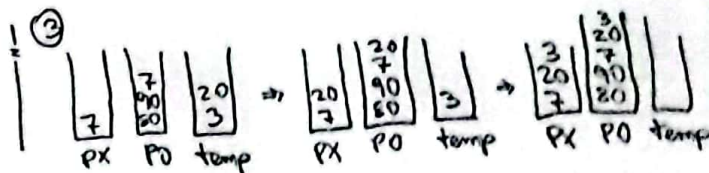
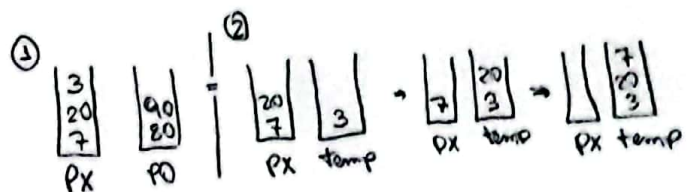
F1: {12, 4, 0, 4, 6, 8, 34, 0, 44, 22, 6, 0}

F2: {5, 7, 67, 23, 5, 33}

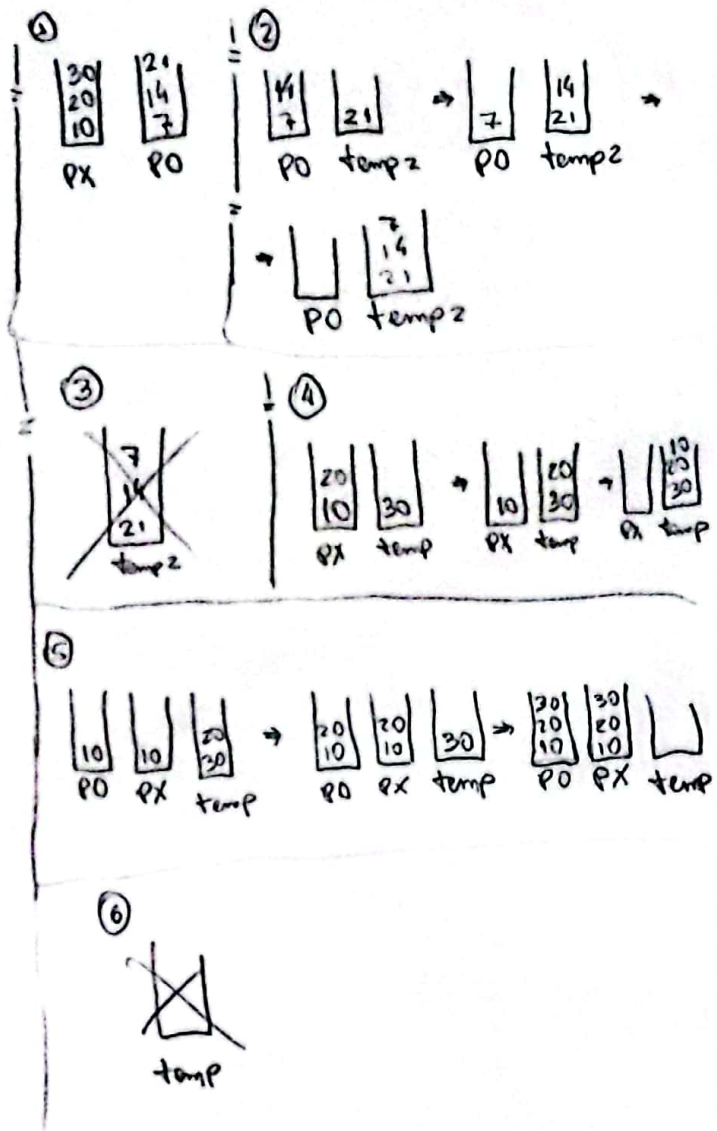
③

```

i) PX = create_pilha();
   PO = create_pilha();
   temp = create_pilha();
   preencher_pilha(PX); preencher_pilha(PO); ①
   enquanto não empty_pilha(PX) fazer
       x = pop(PX);
       push(x, temp); ②
   fim-fazer
   enquanto não empty_pilha(temp) fazer
       x = pop(temp);
       push(x, PO); ③
   fim-fazer
   Destruir(temp); ④
    
```



②
 ii) $PX = \text{create_pilha}();$
 $PO = \text{create_pilha}();$
 $\text{temp} = \text{create_pilha}();$
 $\text{preencher_pilha}(PX);$
 $\text{preencher_pilha}(PO);$
 $\text{temp2} = \text{create_pilha}();$
 enquanto não empty-pilha(PO) fazer
 $x = \text{pop}(PO);$
 $\text{push}(x, \text{temp2});$
 fim-fazer
 destruir(temp2);
 enquanto não empty-pilha(PX) fazer
 $x = \text{pop}(PX);$
 $\text{push}(x, \text{temp});$
 fim-fazer
 enquanto não empty-pilha(temp) fazer
 $x = \text{pop}(temp);$
 $\text{push}(x, PX);$
 $\text{push}(x, PO);$
 fim-fazer
 destruir(temp);



c) combinar Lista (L1, L2)

$L3 = \text{NULL}$;

enquanto (L1 != NULL e L2 != NULL) fazer

se (L1 -> dado < L2 -> dado) então

inserir (L3, L1 -> dado),

L1 = L1 -> proximo;

senão

inserir (L3, L2 -> dado),

L2 = L2 -> proximo;

fim-se

fim-fazer

enquanto (L1 != NULL) fazer

inserir (L3, L1 -> dado),

L1 = L1 -> proximo;

fim-fazer

enquanto (L2 != NULL) fazer

inserir (L3, L2 -> dado),

L2 = L2 -> proximo;

fim-fazer

ii) combinar Lista (L1, L2),

$L3 = \text{NULL}$;

$\text{temp} = \text{NULL}$;

enquanto (L1 != NULL e L2 != NULL) fazer

se (L1 -> dado < L2 -> dado) então

inserir (L3, L1 -> dado);

$\text{temp} = \text{L1} \rightarrow \text{proximo}$;

free (L1);

L1 = temp;

else

inserir (L3, L2 -> dado);

$\text{temp} = \text{L2} \rightarrow \text{proximo}$;

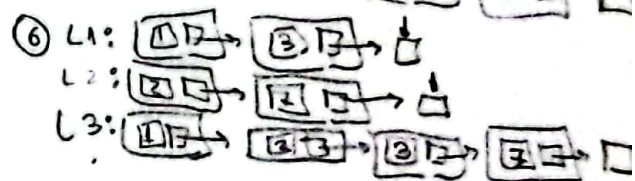
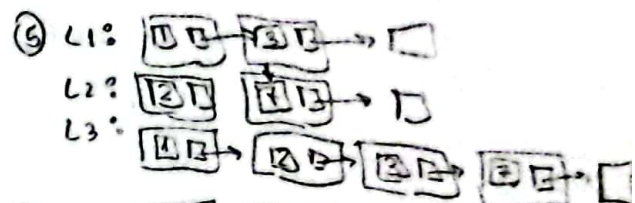
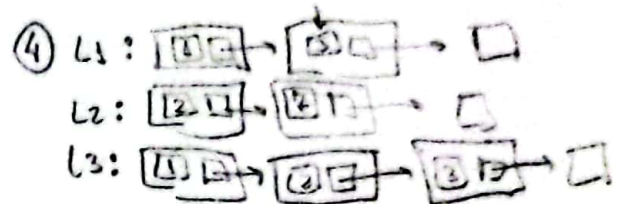
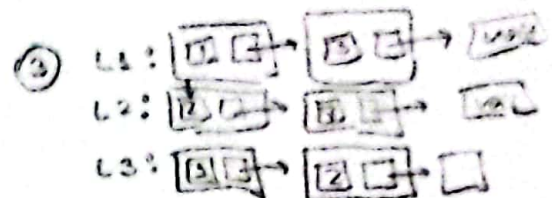
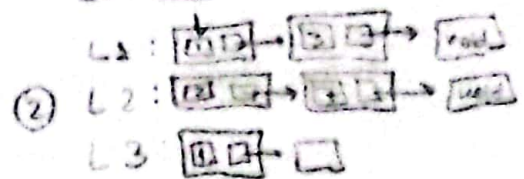
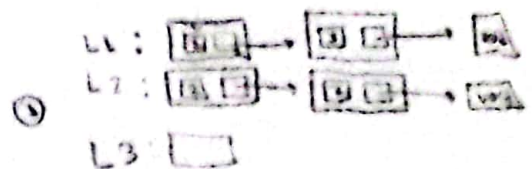
free (L2);

L2 = temp;

fim-se

fim-fazer

④



• continuação

enquanto (L1 != NULL) fazer

inserir (L3, L1 -> dado),

$\text{temp} = \text{L1} \rightarrow \text{proximo}$

free (L1);

L1 = temp;

fim-fazer

enquanto (L2 != NULL) fazer

inserir (L3, L2 -> dado),

$\text{temp} = \text{L2} \rightarrow \text{proximo}$;

free (L2);

L2 = temp;

fim-fazer

5

$X = p \rightarrow \text{seg}$


$q = \text{seg} = x$


$q \rightarrow \text{ont} = p$


$p \rightarrow \text{seg} = q$

$x \rightarrow \text{ont} = q$

4) ii)

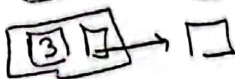
L1: 


L2: 

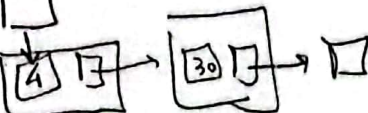
L3: 

L1: 


L2: 


L3: 


L1: 


L2: 


L3: 

L1: 

L2: 

L3: 

L1: 

L2: 

L3: 