Lista de Exercícios

Questão 01

Seu código deve:

1. Implementar a estrutura de dados **lista encadeada simples** incluindo, minimamente, as funções listadas abaixo;
    1. NewList(...) - Inicia uma lista vazia;
    2. Append(...) - Adiciona elemento ao final da lista;
    3. Insert(...) - Adiciona elemento em uma posição específica da lista;
    4. Display(...) - Imprime todos os elementos da lista;
    5. FreeList(...) - Apaga todos elementos da lista;
2. Ler dois vetores e armazená-los em listas encadeadas separadas
    1. Os vetores deverão ser armazenados em suas respectivas listas com "Append(...)";
    2. As listas devem estar ordenadas da mesma forma que os vetores de entrada.
3. Implementar uma função que insere a segunda lista em um determinado ponto da primeira lista;
4. Inserir a segunda lista na primeira e imprimir o resultado;
5. Implementar uma função que remove os valores repetidos de uma lista, mantendo apenas a primeira ocorrência;
6. Remover os valores repetidos da junção das listas e imprimir o resultado.

Seu código deve ler o seguinte modelo de entrada:

```
3                    // Quantidade de linhas a seguir.
0 1 2 3 4 5
0 9 8 0 0
4                    // Posição para inserção.
```

Seu código deve ter o seguinte modelo de saída:

```
0 1 2 3 0 9 8 0 0 4 5      // Print após inserção
0 1 2 3 9 8 4 5            // Print após remoção de valores repetidos
```

Questão 02

Seu código deve:

1. Implementar a estrutura de dados **lista duplamente encadeada não-circular** incluindo, minimamente, as funções listadas abaixo;
    1. NewList(...) - Inicia uma lista vazia;
    2. Append(...) - Adiciona elemento ao final da lista;
    3. Display(...) - Imprime todos os elementos da lista;
    4. FreeList(...) - Apaga todos elementos da lista;
2. Ler dois vetores e armazená-los em listas encadeadas separadas
    1. Os vetores deverão ser armazenados em suas respectivas listas com "Append(...)";
    2. As listas devem estar ordenadas da mesma forma que os vetores de entrada.
3. Implementar uma função que procura os elementos da segunda lista na primeira e retorna uma terceira lista composta pelo caminho percorrido nessa procura;
    1. Assuma que a primeira lista sempre estará ordenada;
    2. Seu algoritmo deve procurar através do melhor caminho sempre;
    3. Você deve armazenar os valores toda vez que houver uma comparação;
4. Você dever imprimir a lista resultante com o caminho percorrido.

Seu código deve ler o seguinte modelo de entrada:

```
2              // Quantidade de linhas a seguir.
3 4 5 7 8
1 10 6 4
```

Seu código deve ter o seguinte modelo de saída:

```
3 3 4 5 7 8 8 7 5 5 4
```

Questão 03 - A turma de estrutura de dados decidiu se reunir no final do semestre para se conhecer. Como muitos gostavam de futebol, montaram dois times com 8 alunos cada e começaram a jogar. Um aluno que chegou atrasado para a reunião gritou: "20 minutos, 3 gols!". Após 20 minutos, o time perdedor teve que escolher alguém para sair. Como o assunto de lista ligadas ainda estava fresco na memória, decidiram fazer um círculo e utilizar esse conceito. Cada um falou um número de 0 a 10, somaram todos eles e escolheram alguém para iniciar a contagem no sentido horário. A partir desse estudante,contaram até chegar na soma total. A pessoa em que a contagem acabar será

quem dará lugar ao estudante que chegou depois.

Seu código deve:

1. Implementar a estrutura de dados **lista encadeada circular** incluindo, minimamente, as funções listadas abaixo;
    1. NewList(...) - Inicia uma lista vazia;
    2. Append(...) - Adiciona elemento ao final da lista;
    3. FreeList(...) - Apaga todos elementos da lista.
2. Sua lista deve armazenar o nome e o valor que cada integrante falou;
    1. O vetor deve ser armazenado com "Append(...)";
    2. A lista deve estar ordenada da mesma forma que o vetor de entrada.
3. Implementar uma função que percorre uma lista encadeada circular;
    1. Inicie a contagem em 0;
    2. Percorra a lista até alcançar o valor total;
    3. Imprima o nome da pessoa escolhida.

Seu código deve ler o seguinte modelo de entrada:

```
9                   // Quantidade de linhas a seguir
Ricardo 3
Ariel 0
Izak 2
Jean 3
Julia 5
Mateus 8
Felipe 1
Luiz 2
Ricardo             // Pessoa escolhida para iniciar a contagem
```

Seu código deve ter o seguinte modelo de saída:

```
Ricardo
```

Questão 04

Universidade Federal da Bahia – UFBA
Instituto de Computação – IC
Departamento de Ciência da Computação – DCC
MATA40 – Estrutura de Dados e Aplicações

Seu código deve:

1. Implementar a estrutura de dados **lista encadeada circular** incluindo, minimamente, as funções listadas abaixo;
    1. NewList(...) - Inicia uma lista vazia;
    2. Append(...) - Adiciona elemento ao final da lista;
    3. Pop(...) - Remove e retorna um elemento da lista;
    4. FreeList(...) - Apaga todos elementos da lista;
2. Implementar a estrutura de dados **pilha** incluindo, minimamente, as funções listadas abaixo;
    1. NewStack(...) - Inicia uma pilha vazia;
    2. Push(...) - Adiciona elemento no topo da pilha;
    3. Pop(...) - Remove e retorna o elemento do topo da pilha;
    4. Display(...) - Imprime todos os elementos da pilha;
    5. FreeStack(...) - Apaga todos elementos da pilha **utilizando pop**;
3. Ler um vetor e armazená-lo em uma lista encadeada circular;
    1. O vetor deve ser armazenado com "Append(...)";
    2. A lista deve estar ordenada da mesma forma que o vetor de entrada.
4. Implementar uma função que procura o próximo maior elemento da lista, remove-o e o adiciona a uma pilha que deve ser retornada;
    1. A primeira pesquisa deve começar no primeiro valor da lista (o primeiro valor do vetor recebido);
    2. Ao chegar ao final da lista, você deve continuar até encontrar o valor novamente e removê-lo;
    3. A partir desse momento, as novas pesquisas não necessariamente iniciarão no primeiro valor, mas sim de onde você removeu o último;
    4. Você deve contar a quantidade de voltas feitas na lista e imprimir esse valor quando a lista estivar vazia;
    5. Você deve retornar a pilha resultante (que estará ordenada com os menores valores no topo e maiores na base);
5. Você dever imprimir a pilha resultante.

Seu código deve ler o seguinte modelo de entrada:

```
1                  // Quantidade de linhas a seguir.
5 0 7 2 4
```

Seu código deve ter o seguinte modelo de saída:

```
5                  // Quantidade de voltas realizadas
0 2 4 5 7          // Valores nas ordem armazenada na pilha.
```

Questão 05

Seu código deve:

1. Implementar a estrutura de dados **pilha** incluindo, minimamente, as funções listadas abaixo;
    1. NewStack(...) - Inicia uma pilha vazia;
    2. Push(...) - Adiciona elemento no topo da pilha;
    3. Pop(...) - Remove e retorna o elemento do topo da pilha;
    4. Display(...) - Imprime todos os elementos da pilha;
    5. FreeStack(...) - Apaga todos elementos da pilha **utilizando pop**;
2. Ler dois vetores e armazená-los em pilhas separadas
    1. Os vetores deverão ser armazenados em suas respectivas pilhas com "Push(...)";
    2. As pilhas resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo).
3. Implementar uma função que empilhe a segunda pilha na primeira sem alterar a ordem das mesmas;
    1. Utilize apenas pilhas.
4. Você dever imprimir a pilha resultante.

Seu código deve ler o seguinte modelo de entrada:

```
2              // Quantidade de linhas a seguir.
1 2 3 4 5
6 7 8 9 0
```

Seu código deve ter o seguinte modelo de saída:

```
0
9
8
7
6
5
4
3
2
1
```

Questão 06

Seu código deve:

1. Implementar a estrutura de dados **pilha** incluindo, minimamente, as funções listadas abaixo;
    1. NewStack(...) - Inicia uma pilha vazia;
    2. Push(...) - Adiciona elemento no topo da pilha;
    3. Pop(...) - Remove e retorna o elemento do topo da pilha;
    4. FreeStack(...) - Apaga todos elementos da pilha **utilizando pop**;
2. Ler um vetor e armazená-lo em uma pilha;
    1. O vetor deve ser armazenado com "Push(...)";
    2. A pilha resultante deve estar ordenada da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo).
3. Implementar uma função que utilize 3 pilhas e transfira os elementos da primeira pilha para a terceira utilizando a segunda como auxiliar;
    1. Utilize apenas pilhas.
    2. A primeira pilha sempre estará ordenada (maior valor na base, menor no topo);
    3. Você nunca deve empilhar um valor maior sobre um valor menor;
    4. Utilize o método mais eficiente (cuidado com plágios);
4. Você dever imprimir quantos passos foram necessários para mover a primeira pilha para a terceira.

Seu código deve ler o seguinte modelo de entrada:

```
1       // Quantidade de linhas a seguir
2 1
```

O que seu código deve simular:

```
1   |   |
2   |   |

1º
|   |   |
2   1   |

2º
|   |   |
|   1   2

3º
|   |   1
|   |   2
```

Seu código deve ter o seguinte modelo de saída:

```
3
```

Questão 07

Seu código deve:

1. Implementar a estrutura de dados **pilha** incluindo, minimamente, as funções listadas abaixo;
    1. NewStack(...) - Inicia uma pilha vazia;
    2. Push(...) - Adiciona elemento no topo da pilha;
    3. Pop(...) - Remove e retorna o elemento do topo da pilha;
    4. Display(...) - Imprime todos os elementos da pilha;
    5. FreeStack(...) - Apaga todos elementos da pilha **utilizando pop**;
2. Implementar a estrutura de dados **fila** incluindo, minimamente, as funções listadas abaixo;
    1. NewQueue(...) - Inicia uma fila vazia;
    2. Append(...) - Adiciona elemento no final da fila;
    3. Pop(...) - Remove e retorna o elemento na frente da fila;
    4. Display(...) - Imprime todos os elementos da fila;
    5. FreeQueue(...) - Apaga todos elementos da fila **utilizando pop**;
3. Ler dois vetores e armazená-los em pilhas separadas
    1. Os vetores deverão ser armazenados em suas respectivas pilhas com "Push(...)";
    2. As pilhas resultantes resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo).
    3. Imprima as pilhas.
4. Transferir os elementos da pilha para filas separadas
    1. Os vetores deverão ser armazenados em suas respectivas filas com "Append(...)";
    2. As filas resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final);
    3. Utilize pilhas para alcançar isso (dica: simule uma fila utilizando pilhas);
    4. Imprima as filas.
5. Implementar uma função que recebe 2 filas e retorne uma terceira fila que intercala os valores das duas primeiras.
    1. Utilize apenas filas;
    2. Valores da primeira fila primeiro;
    3. A fila maior terá seus últimos elementos no final;
    4. Imprima a fila resultante;

Seu código deve ler o seguinte modelo de entrada:

```
2                    // Quantidade de linhas a seguir.
0 2 4 6 8 10 12
1 3 5 7 9
```

Seu código deve ter o seguinte modelo de saída:

```
0 2 4 6 8 10 12
1 3 5 7 9
0 2 4 6 8 10 12
1 3 5 7 9
0 1 2 3 4 5 6 7 8 9 10 12
```

Questão 08

Na última semana, a fila de entrada de um certo evento ficou extraordinariamente grande. Os organizadores perceberam que algumas pessoas necessárias para evento acontecer estavam presas nessa fila. Além disso, perceberam também uma certa quantidade de idosos e grávidas ao longo da mesma.

Para evitar que essa situação se repita, a equipe que organiza eventos como esse decidiu pedir à turma de estrutura de dados da UFBA para escrever um programa para auxiliá-los. Eles preferiram que esse programa fosse acionado manualmente toda vez que eles julgassem que a fila precisasse ser organizada. O programa deve dividir a fila em duas:

1. Não prioridade.
2. Prioridade: a partir de 60 anos, até 5 anos, grávidas e palestrantes.
    - Grávidas e palestrantes serão especificados pela posição na fila.

Seu código deve:

1. Implementar a estrutura de dados **fila** incluindo, minimamente, as funções listadas abaixo;
    1. NewQueue(...) - Inicia uma fila vazia;
    2. Append(...) - Adiciona elemento no final da fila;
    3. Pop(...) - Remove e retorna o elemento na frente da fila;
    4. Display(...) - Imprime todos os elementos da fila;
    5. FreeQueue(...) - Apaga todos elementos da fila **utilizando pop**;
2. Ler um vetor e armazená-lo em uma **fila**;
    1. O vetor deve ser armazenado na **fila** com "Append(...)";
    2. A **fila** resultante deve estar ordenada da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final);
3. Transferir os elementos dentro das especificações para uma **fila** separada;
    1. Todas as operações de transferência de **fila** deverão ser realizadas utilizando filas: Pop(...)" para remover da **fila** inicial, "Append(...)" para adicionar na **fila** auxiliar/final;
    2. As filas resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final);
    3. Imprima as filas.

Seu código deve ler o seguinte modelo de entrada:

```
2                    // Quantidade de linhas a seguir.
50 65 30 33 11 3 25 39 17 22 27 26 44 28 53 28 20 70 58 27 33 36 49 17    // Vetor de idades
7 8 14   // Vetor com a posição das grávidas e palestrantes na fila
```

Seu código deve ter o seguinte modelo de saída:

```
50 30 33 11 17 22 27 26 44 53 28 20 58 27 33 36 49 17     // Fila original sem os preferenciais
65 3 25 39 28 70        // Fila preferencial
```

Questão 09

Seu código deve:

1. Implementar a estrutura de dados **pilha** incluindo, minimamente, as funções listadas abaixo;
    1. NewStack(...) - Inicia uma pilha vazia;
    2. Push(...) - Adiciona elemento no topo da pilha;
    3. Pop(...) - Remove e retorna o elemento do topo da pilha;
    4. Display(...) - Imprime todos os elementos da pilha;
    5. FreeStack(...) - Apaga todos elementos da pilha **utilizando pop**;
2. Implementar a estrutura de dados **fila** incluindo, minimamente, as funções listadas abaixo;
    1. NewQueue(...) - Inicia uma fila vazia;
    2. Append(...) - Adiciona elemento no final da fila;
    3. Pop(...) - Remove e retorna o elemento na frente da fila;
    4. Display(...) - Imprime todos os elementos da fila;
    5. FreeQueue(...) - Apaga todos elementos da fila **utilizando pop**;
3. Implementar a estrutura de dados **árvore de grau 3** incluindo, minimamente, as funções listadas abaixo;
    1. NewTree(...) - Inicia uma árvore vazia;
    2. Add(...) - Adiciona o próximo elemento na árvore;
    3. RemoveLeftmostLeaf(...) - Remove a folha mais à esquerda da árvore e retorna seu conteúdo;
    4. DisplayAndClearTree(...) - Imprime todos os elementos da árvore utilizando "RemoveLeftmostLeaf(...)" até a árvore ser esvaziada (ao remover um elemento, o ponteiro do nó pai passa a ser NULL e nenhuma operação de reorganização da árvore é necessária);
4. Ler um vetor e armazenar uma cópia do mesmo em uma pilha e uma cópia em uma fila;
    1. Os vetores deverão ser armazenados em suas respectivas estruturas com seu método de adicionar associado;
    2. As estruturas devem estar ordenadas da mesma forma que os vetores de entrada (primeiro/base/frente → último/topo/final).
5. Implementar uma função que recebe uma fila, uma pilha e cria/manipula duas árvores de grau 3 da seguinte maneira:
    1. Ambas devem ser preenchidas da esquerda para a direita, completando um nível por vez.
    2. A primeira deve ser preenchida com a fila, a segunda com a pilha.
    3. Limpe as árvores da esquerda para a direita, priorizando sempre remover a folha mais profunda à esquerda (RemoveLeftmostLeaf).
    4. Imprima o conteúdo à medida que o remove (DisplayAndClearTree + RemoveLeftmostLeaf).

Seu código deve ler o seguinte modelo de entrada:

```
1           // Quantidade de linhas a seguir.
1 2 3 4 5 6 7 8 9  // Vetor que deve ser transformado em uma fila e em uma pilha
```

Processo:

```
|           Fila -> Arvore            |          Pilha -> Arvore            |
|(Frente) 1 2 3 4 5 6 7 8 9 (Fundo)   | (Base) 1 2 3 4 5 6 7 8 9 (Topo)     |
|                                     |                                     |
|              _____1_____            |              _____9_____            |
|             /     |     \           |             /     |     \           |
|            2      3      4          |            8      7      6          |
|           /|\    /|                 |           /|\    /|                 |
|          5 6 7  8 9                 |          5 4 3  2 1                 |
```

Seu código deve ter o seguinte modelo de saída:

```
5 6 7 2 8 9 3 4 1    // Saída obtida ao esvaziar a árvore criada com a fila.
5 4 3 8 2 1 7 6 9    // Saída obtida ao esvaziar a árvore criada com a pilha.
```

```
^               ^
|               |_ Último elemento removido é a raiz.
|_ Primeiro elemento removido é o mais à esquerda.
```

Questão 10

Seu código deve:

1. Implementar a estrutura de dados **árvore binária completa** incluindo, minimamente, as funções listadas abaixo;
    1. NewTree(...) - Inicia uma árvore vazia;
    2. Add(...) - Adiciona o próximo elemento na árvore;
    3. ClearTree(...) - Esvaziada a árvore;
2. Ler um vetor e armazená-lo em uma árvore binária completa;
    1. Ele deve ser armazenado com "Add(...)";
    2. Todos os nós de um dado nível devem ser preenchidos (da esquerda para a direita) antes de preencher o nível seguinte;
    3. Os elementos devem ser preenchidos na mesma ordem do vetor de entrada;
3. Implementar uma função que recebe uma árvore e dois números para pesquisa.
    1. Considere que não haverá valores repetidos;
    2. Caso ambos elementos sejam encontrados, imprima a diferença de nível deles;
    3. Caso apenas um seja encontrado, imprima a altura do mesmo;
    4. Caso nenhum seja encontrado, imprima a profundidade da árvore.
4. Seu código deve receber uma árvore e "N = linhas - 1" pares de valores para pesquisa.

Seu código deve ler o seguinte modelo de entrada:

```
5                          // Quantidade de linhas
1 2 3 4 5 6 7 8 9 10       // Vetor de entrada para a árvore
1 2                        // Pares de valores para pesquisa
1 11
3 10
0 15
```

```
Árvore Binária Completa:
|                 ____1____              |
|               /          \             |
|              2            3            |
|           /     \     /     \          |
|          4       5   6       7         |
|        /   \    /                      |
|       8     9  10                      |
```

Seu código deve ter o seguinte modelo de saída:

```
1
3
2
3
```