Universidade Federal de Uberlândia
Curso de Graduação em Gestão da Informação
3º Trabalho de Estruturas de Dados I – Prof. Daniel A. Furtado
Trabalho Individual – Filas e Listas em Array

INTRUÇÕES GERAIS – LER ATENTAMENTE

- Esta atividade deve ser realizada individualmente;
- Este trabalho deve ser feito utilizando a linguagem C# ou Python. Trabalhos implementados utilizando outras linguagens serão anulados;
- **Todo o código deve ser colocado em um único arquivo**, pois apenas um arquivo (.cs ou .py) deve ser entregue. Não crie arquivos adicionais para as classes (o único arquivo a ser entregue deve conter todas as definições de classes, assim como o programa principal).
- **Não compactar** os arquivos no momento da entrega;
- Os recursos adequados das linguagens devem ser utilizados;
- Esteja atento às **observações sobre plágio** apresentadas no final desde documento;
- Trabalhos com implementações utilizando trechos de códigos retirados de sites da Internet, de trabalhos de colegas de classe ou de semestres anteriores serão anulados;
- As implementações não devem conter qualquer conteúdo de caráter imoral, desrespeitoso, pornográfico, discurso de ódio, desacato, etc.;
- O trabalho deve ser entregue até a data definida pelo professor em aula síncrona pelo Sistema de Aplicação de Testes (SAAT);
- Trabalhos enviados por e-mail ou por outros meios digitais (como o MS Teams) **não serão considerados** (veja instruções no final);

Parte 1

Uma **Deque** é uma estrutura de dados linear que permite a inserção e a remoção de elementos nas duas extremidades. Por ter essa característica, uma Deque pode ser utilizada para atuar tanto como uma pilha quanto como uma fila, desde que apenas as operações em questão sejam utilizadas. Utilize a lógica de *array* circular proposta nos slides de aula (Módulo 3 - Filas) em conjunto com os "ponteiros" **Início** e **Final** para implementar a estrutura **Deque** como um tipo abstrato de dados. A classe deve ter os seguintes métodos públicos:

- `AddRear` – para adicionar um novo elemento no final da estrutura
- `RemoveRear` – para remover e **retornar** o elemento do final da estrutura
- `AddFront` – para adicionar um novo elemento no início da estrutura
- `RemoveFront` – para remover e **retornar** o elemento do início da estrutura

Dicas: na implementação das operações **AddRear** e **RemoveFront**, os ponteiros precisam avançar no sentido **horário** utilizando incremento com módulo, de forma similar às operações *Enqueue* e *Dequeue*. Nas operações **AddFront** e **RemoveRear**, os ponteiros precisam avançar no sentido **anti-horário**. Neste caso, quando atingirem a posição 0, a próxima atualização anti-horária deve coloca-los na posição **n-1** do *array*, onde **n** é o tamanho do *array*.

Após implementação da classe referente ao TAD Deque, mostre, no programa principal, um exemplo utilizando o TAD e seus métodos.

Parte 2

Implementar o TAD **Lista** em a*rray* utilizando as definições de classe e métodos apresentadas nos slides de aula "Módulo 4 – Listas". Devem ser implementadas todas as principais operações indicadas no material, incluindo aquelas cujo código foi apresentado e também aquelas cujo código não foi explicitamente mostrado no material:

- **AddFirst – adiciona um elemento no início da lista**
- **AddLast – adiciona um elemento no final da lista**
- **InsertAt(i, item) – insere o elemento item na posição i da lista**
- **RemoveFirst – remove o primeiro elemento da lista**
- **RemoveLast – remove o último elemento da lista**
- **Remove(item) – remove a primeira ocorrência do elemento item**
- **RemoveAt(i) – remove o elemento na posição i da lista**
- **Get(i) – retorna o elemento na posição i da lista, sem remover**
- **Set(i , newItem) – substitui o item na posição i da lista por newItem**
- **Find(item) – retorna a posição da primeira ocorrência do item (ou -1)**

Implementar também o método **Display**() para apresentar todos os elementos da lista. Em seguida, crie um código principal para exemplificar o uso do TAD Lista. O código deve inserir e remover elementos usando todos os métodos, apresentar a lista, etc.

Acrescente o método **AddSorted**() para inserção ordenada. Crie uma segunda lista no programa principal e faça inserções/apresentações para testar o método.

OBS: Ao término do trabalho o programa principal deve conter todos os trechos de código de teste, na sequência das partes do exercício. Ao ser executado, primeiramente o código deve mostrar os testes relacionados ao TAD Deque, em seguida deve mostrar os testes relacionados ao TAD Lista e por fim os testes relacionados à lista com os elementos ordenados.

Entrega

Um único arquivo (.cs ou .py) contendo todas as definições de classes e também o programa principal deve ser enviado pelo Sistema de Aplicação de Testes (SAAT) até a data limite

indicada pelo professor. Não envie arquivos compactados ou outros arquivos do Visual Studio que não sejam o arquivo de código principal.

Sobre Eventuais Plágios

Este é um trabalho individual. Os alunos envolvidos em qualquer tipo de plágio, total ou parcial, seja entre equipes ou de trabalhos de semestres anteriores ou de materiais disponíveis na Internet (exceto os materiais de aula disponibilizados pelo professor), serão duramente penalizados (art. 196 do Regimento Geral da UFU). Todos os alunos envolvidos terão seus **trabalhos anulados** e receberão **nota zero**.