CMP1054 - Estrutura de Dados I
5ª Lista de Exercícios - Filas
Max Gontijo de Oliveira

- Todos os métodos e funções criados nas questões deverão ser testadas em um programa principal (`main`).
- Caso haja necessidade, crie parâmetros adicionais para os métodos ou funções recursivos além dos explicitamente solicitados nas questões.
- Quando não estiver explícito na questão, o aluno poderá escolher entre criar um algoritmo iterativo ou recursivo.
- Em todas as questões, considere as classes `Node` e `Fila` conforme visto em sala de aula.

```
class Node {
   public:
      TIPO x;
      Node *prox;
};

class Fila {
        private:
                Node *inicio;
                Node *fim;

        public:
                Fila(); // Inicializa inicio e fim com NULL
                void queue(TIPO); // enfileirar
                TIPO dequeue();   // desenfileirar
                TIPO first();     // elemento do início da fila
                TIPO last();      // elemento do fim da fila
                bool is_empty();  // verifica se fila está vazia
                void clear();     // limpar fila
                int size();       // tamanho da fila
                void inverter();  // inverte ordem dos elementos da fila
                void print();     // imprimir elementos da fila na tela

                // Outros métodos
};
```

- Na classe `Node` acima, `TIPO` se refere ao tipo de dado que a fila deverá armazenar (`int`, `char`, `float`). Todavia, NÃO é obrigatório o uso de `template`. Assim, se o aluno preferir, pode criar uma classe `Node` e uma classe `Fila` para cada questão.
- Considere ainda a existência da classe `Pilha` conforme cobrada na lista de exercícios anterior.

1. Crie um MÉTODO chamado `queue` na classe `Fila` que receba um `int` e guarde esse valor no fim da fila.
2. Crie um MÉTODO chamado `dequeue` na classe `Fila` cujo retorno seja do tipo `int`; O método deverá retornar o valor do início da fila, removendo o respectivo `Node` da memória. Caso a fila esteja vazia, o método deverá retornar 0.
3. Crie um MÉTODO chamado `first` na classe `Fila` cujo retorno seja do tipo `int`; O método deverá retornar o valor do início da fila sem remover o respectivo `Node` da memória. Caso a fila esteja vazia, o método deverá retornar 0.
4. Crie um MÉTODO chamado `last` na classe `Fila` cujo retorno seja do tipo `int`; O método deverá retornar o valor do fim da fila sem remover o respectivo `Node` da memória. Caso a fila esteja vazia, o método deverá retornar 0.
5. Crie um MÉTODO chamado `is_empty` na classe `Fila` que retorne `true` se a fila estiver vazia ou `false` caso contrário.
6. Crie um MÉTODO chamado `clear` na classe `Fila` que remova todos os elementos da fila.
7. Crie um MÉTODO chamado `size` na classe `Fila` que retorne a quantidade de elementos da fila.
8. Crie um MÉTODO chamado `inverter` na classe `Fila` para alterar a ordem dos elementos da fila. Dentro do método, instancie e utilize uma `Pilha` para realizar essa mudança. Utilize apenas os métodos `push`, `pop` e `is_empty` do objeto `Pilha` e os métodos `queue`, `dequeue` e `is_empty` da própria `Fila`. NÃO é um método para imprimir os elementos. É um método para, efetivamente, inverter a ordem dos elementos na fila.
9. Crie um MÉTODO chamado `print` na classe `Fila` para imprimir todos os elementos da fila em ordem de inserção.
10. Questão para testar sua fila: crie um PROGRAMA que instancie uma classe `Fila` (essa classe deve ter os métodos criados nas questões 1, 2, 3, 4, 5, 6, 7, 8 e 9) e em seguida, execute os seguintes passos:

1. *Programa lê do teclado um número inteiro OP.*
   (a) *Se $OP = 1$ o programa deverá ler um inteiro X e adicionar esse valor no fim da fila (`add`).*
   (b) *Se $OP = 2$ o programa deverá remover o elemento do início da fila e exibir o valor na tela (`remove`). Caso a fila esteja vazia, não tente remover: escreva uma mensagem na tela informando.*
   (c) *Se $OP = 3$ o programa deverá exibir na tela o valor do elemento do início da fila (`first`).*
   (d) *Se $OP = 4$ o programa deverá exibir na tela o valor do elemento do fim da fila (`last`).*
   (e) *Se $OP = 5$ o programa deverá limpar a fila (`clear`).*
   (f) *Se $OP = 6$ o programa deverá exibir a quantidade de elementos armazenados na fila (`size`).*
   (g) *Se $OP = 7$ o programa deverá inverter a ordem dos elementos da fila (`inverter`).*
   (h) *Se $OP = 8$ o programa deverá exibir todos os elementos da fila sem removê-los (`print`).*
   (i) *Se $OP = 0$ o programa deverá finalizar.*
   (j) *Se OP for qualquer outro valor, deverá ser ignorado.*
2. *Após a execução da operação determinada por OP, se $OP = 0$, então o programa deverá finalizar; caso contrário, o programa voltará a execução no passo 1.*

11. Em um grafo direcionado as arestas indicam não apenas os vértices envolvidos em uma ligação mas também qual é o vértice de origem e qual é o vértice de destino. Uma matriz A de dimensões $N \times N$ pode ser utilizada para representar as arestas de um grafo com N vértices. Assim, A_{ij} será igual a 1 caso exista uma aresta que parta do vértice i e vá para o vértice j; ou A_{ij} será igual a 0 caso não exista uma aresta que parta do vértice i e vá para o vértice j.

    A menor distância entre dois vértices quaisquer é determinada pela quantidade mínima de arestas que devem ser atravessadas entre o vértice de origem e o vértice de destino.

    Crie uma FUNÇÃO que receba por parâmetro, o ponteiro de uma matriz que representa um grafo direcionado, um inteiro que indica a quantidade de vértices, e um inteiro que represente o vértice de origem. A função deverá retornar um vetor com a quantidade de elementos determinada pela quantidade de vértices onde cada elemento j do vetor terá a menor distância entre o vértice passado por parâmetro e o vértice j. Considere que quando j for igual ao vértice de origem, então a distância será 0. Caso não haja caminho entre o vértice de origem e o vértice j, essa posição no vetor deverá receber um valor bem alto (recomenda-se a quantidade de vértices pois nenhuma distância mínima passa por um vértice mais de uma vez).

12. Filas são muito utilizadas em controle de fluxos (tanto em nível de hardware quanto em nível de software). Parte do controle de fluxo está em multiplexar diversos conjuntos de dados de diferentes fluxos em um único canal em um determinado momento.

    Crie um PROGRAMA que simule a multiplexação de três fluxos contínuos de dados que vão utilizar um único canal (por exemplo, três programas que querem utilizar a placa de rede do computador). Considere que o algoritmo de prioridade de uso do canal é FIFO. Considere ainda que quando os fluxos estão enviando dados continuamente, são enfileirados no canal, o primeiro a chegar de cada fluxo, alternando entre os fluxos.

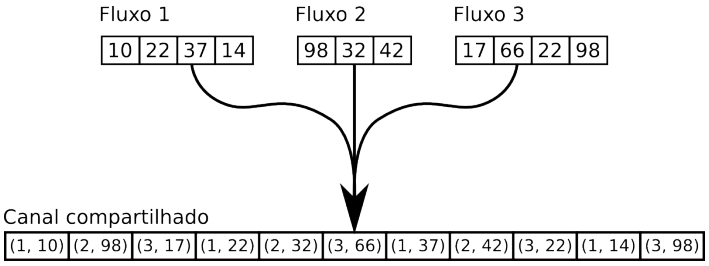
    Para efetuar a simulação, seu programa deverá passar pelos seguintes passos:

    1. *Programa lê do teclado diversos números inteiros que são enfileirados em uma fila de inteiros que representa o primeiro fluxo de dados.*
    2. *Programa lê do teclado diversos números inteiros que são enfileirados em uma outra fila de inteiros que representa o segundo fluxo de dados.*
    3. *Programa lê do teclado diversos números inteiros que são enfileirados em uma outra fila de inteiros que representa o terceiro fluxo de dados.*
    4. *Programa imprime na tela os elementos das três filas para que possamos visualizar o estado dos fluxos.*
    5. *Programa cria uma outra fila (que representará o canal compartilhado). Essa fila, porém, não será apenas uma fila de inteiros, mas uma fila de objetos que armazenam dois inteiros (um para armazenar o valor que vem dos fluxos e outro para armazenar o identificador do fluxo).*
    6. *Programa alterna entre os fluxos desenfileirando um a um os valores. A cada iteração, o programa deve desenfileirar um valor de um dos fluxos e enfileirar esse valor e o identificador do fluxo na fila do canal. Na iteração seguinte, deverá fazer o mesmo com outro fluxo. E assim será até que todos os fluxos estejam vazios.*
       - *A medida em que um fluxo se esgota, não haverá mais desenfileiramento nele.*
    7. *Ao final, deverá ser impresso na tela os elementos da fila do canal (identificador e valor).*

    Dica 1: nos passos 1, 2 e 3, as leituras e inserções nas filas terminarão quando o usuário digitar o número -1.

    Dica 2: utilize uma lista circular de objetos `Fila` para realizar a alternância entre os fluxos de dados.

    A figura a seguir demonstra o processo:



13. Outra aplicação de filas em relação à controle de fluxos é o processo de demultiplexação de dados em um canal. Nesse caso, o receptor gerenciador de um determinado canal deveria distribuir os dados que chegam em um canal compartilhado para os devidos destinos (por exemplo, uma placa de rede que recebe pacotes endereçados a diversas aplicações).

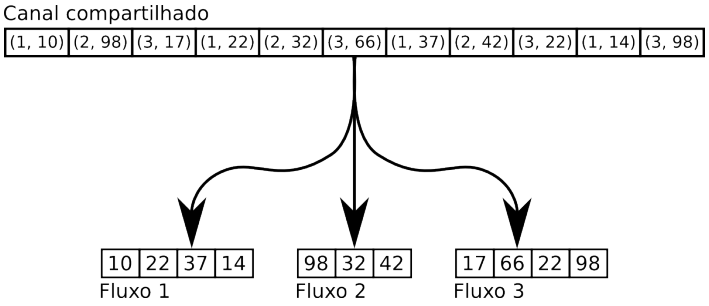
    Crie um PROGRAMA que simule a demultiplexação de dados de um canal e entregue esses dados aos destinos (fluxos representados por objetos `Fila`).

    Para efetuar a simulação, seu programa deverá passar pelos seguintes passos:

    1. *Programa lê do teclado diversos pares de números inteiros (um com o identificador de um dos três destinos e outro um valor qualquer). Esses pares serão enfileirados em uma fila de objetos que armazene esses dois números. Essa fila irá representar o canal compartilhado.*
    2. *Programa imprime todos os elementos da fila do canal.*
    3. *Programa cria outras três filas que representarão os fluxos de destino possíveis.*
    4. *Programa desenfileira todos os elementos da fila do canal, um a um. Para cada elemento desenfileirado, o seu valor deverá ser enfileirado em uma das outras três filas de inteiros que representam os fluxos de destino, de modo que a escolha do fluxo se dará pela identificação do elemento.*
    5. *Ao final, deverão ser impressos os itens das três filas que representam os fluxos.*

    Dica: No passo 1, as leituras e inserções na fila terminarão quando o usuário digitar o identificador -1.

    A figura a seguir demonstra o processo:



14. Considere a existência de um labirinto. Considere que um robô seja colocado em algum lugar desse labirinto. Considere ainda que esse robô tem o mapa completo do labirinto. Crie um PROGRAMA que seja capaz de dar ao robô, o menor caminho entre sua localização atual e a saída do labirinto.

    Seu programa deverá ser capaz de ler um arquivo contendo os dados de entrada. Segue abaixo o layout do arquivo de entrada.

```
1 Altura do labirinto (int)
2 Largura do labirinto (int)
3 Localização inicial do robô - linha (int)
4 Localização inicial do robô - coluna (int)
5 Início das linhas que descrevem o mapa do labirinto (char)
```

    Segue um exemplo de arquivo:

```
11
30
9
1
******************************
*                            *
* ******************** *****  *
* **                   **  *  *
* ******************** **  *  *
* **                 * **  *  *
* ** ***** ******** * **  *  *
* ** *   * *      ** *     *  *
* ** ***** * ****** *******  *
* ** *     *                  *
****@**************************
```

No arquivo, o mapa do labirinto será composto dos caracteres ‘ ’ (espaço), ‘*’ e ‘@’. Os caracteres ‘ ’ determinam por onde o robô é permitido caminhar; os caracteres ‘*’ determinam paredes intransponíveis pelo robô; e caractere ‘@’ determina a saída do labirinto.

No exemplo acima, o mapa tem 11 linhas e 30 colunas e o robô inicializa na linha 9/coluna 1.

O seu programa deverá imprimir o mapa mostrando o robô na posição inicial por meio de um caractere ‘R’ e utilizar caracteres ‘x’ para indicar o caminho percorrido. Veja o resultado para o exemplo acima:

```
******************************
*xxxxxxxxxxxxxxxxxxxxxxxxxxxx*
*x******************* *******x*
*x***                **  ***x*
*x******************* **  ***x*
*x***xxxxxxxxxxxxxxxxx* **  ***x*
*x*********  *********x* **  ***x*
*x****x    * *      ***x*     ***x*
*x*****x***** * ******x*********x*
*R***x*      *        xxxxxxxxxx*
****@**************************
```

Caso não exista saída do labirinto, seu programa deverá escrever essa informação na tela.

Dica: Para realizar a leitura do arquivo, você pode utilizar a classe `ifstream`. Para isso, você precisa de:

1. Importar a biblioteca necessária: `#include <fstream>`
2. Abrir o arquivo: `ifstream entrada("arquivo_labirinto.txt");`
3. Obtenha a quantidade de linhas e armazene-a em um inteiro: `entrada » lins;`
4. Obtenha a quantidade de colunas e armazene-a em um inteiro: `entrada » cols;`
5. Obtenha a posição inicial do robô (linha) armazene-a em um inteiro: `entrada » i_robo;`
6. Obtenha a posição inicial do robô (coluna) armazene-a em um inteiro: `entrada » j_robo;`
7. Como a última leitura foi de um valor inteiro e as próximas serão de strings, então limpe o *buffer* do teclado: `entrada.ignore();`
8. Obtenha as demais linhas do arquivo que representam o labirinto, uma a uma: `getline(entrada, linha);`
9. Não se esqueça de fechar o arquivo: `entrada.close();`