UFJF/DCC Estrutura de Dados e Lab. de Programação II – TVC 2 29/10/2018

Nome: _____ Matrícula: _____

Leia atentamente as instruções abaixo:

- A avaliação é **individual** e **sem qualquer tipo de consulta**.
- Acesse o site do curso e faça o download do projeto único do Code::Blocks que contém todos os códigos para desenvolvimento desta avaliação. Descompacte o arquivo em um diretório local da sua máquina para iniciar a avaliação.
- Preencher o cabeçalho do `main.cpp` seu nome completo e seu número de matrícula.
- A resposta de cada questão deve, **obrigatoriamente**, estar entre cada par de marcadores (//Qi, //-Qi). Assim, a questão 1 está entre //Q1 e //-Q1, a questão 2 entre //Q2 e //-Q2 e assim por diante. Não remover, em hipótese alguma, tais marcadores de questões da sua avaliação. Caso sua solução tenha mais de uma função ou operação, elas devem estar entre esses marcadores, obrigatoriamente.
- Ao terminar a avaliação, enviá-la ao servidor antes de sair do laboratório. Ao acessar o servidor com seu número de matrícula, faça o upload de cada arquivo de código que contém as respostas das questões da sua avaliação. Verifique suas respostas enviada e confirme a submissão. **O envio dos códigos ao servidor é de inteira responsabilidade do aluno!** O endereço de acesso ao servidor é: `http://172.18.40.97:8080/edlab2ufjf/`

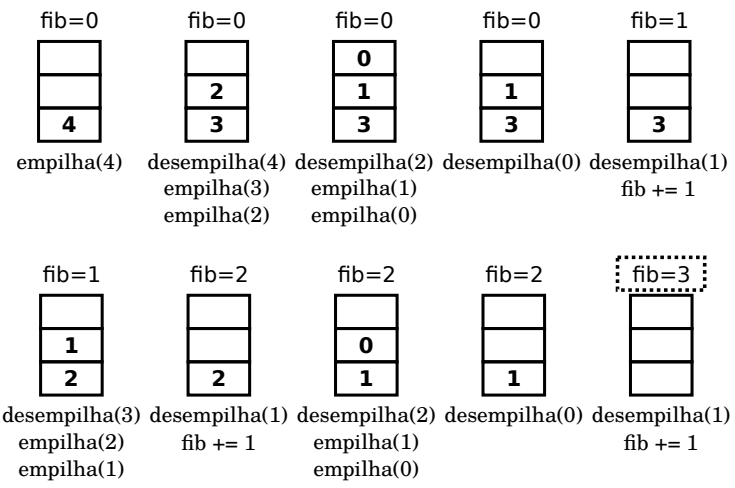
Questões:

1. (25 Pontos) Considere o TAD `PilhaCont` para representar uma **pilha** de inteiros através de um vetor (ver os detalhes deste TAD no projeto). Sabe-se que pilhas podem ser usadas para substituir chamadas recursivas. Sendo assim, implemente no arquivo `main.cpp` a função `void fibonacci(int n)` para calcular o valor do elemento `n` da sequência de *Fibonacci* (sequência em que o próximo elemento é a soma dos dois anteriores).

   A função utiliza um acumulador (`fib`) para guardar o resultado, e começa empilhando o índice inicial `n`. Os passos seguintes são executados iterativamente:

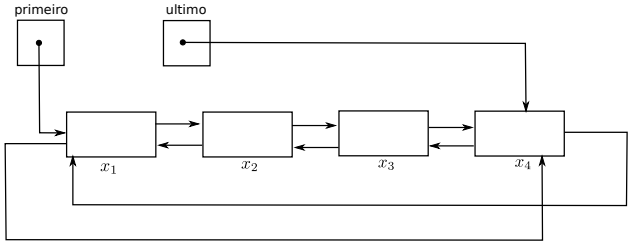
   - Desempilha-se um valor e, para os casos que não sejam base, empilha-se os dois valores necessários para se calcular o elemento desempilhado;
   - Se o valor desempilhado for o caso base 1 então deve-se somar 1 no acumulador, já que `fibonacci(1) = 1`;
   - Se o valor desempilhado for o caso base 0 então não é preciso fazer nada, já que `fibonacci(0) = 0`.

   O acumulador contém o resultado de `fibonacci(n)` quando a pilha ficar vazia. Deve-se exibir uma mensagem de erro e parar o programa se $n < 0$. A imagem abaixo ilustra o processo para `fibonacci(4)`.



2. (25 Pontos) Considere o TAD `ListaEncad` para representar uma **lista simplesmente encadeada** de inteiros (ver os detalhes deste TAD no projeto). Implemente a operação `ListaEncad* ListaEncad::intercala(ListaEncad *lista)` para criar e retornar uma nova lista formada pela intercalação alternada dos nós de duas outras listas: a lista implícita (aquela para a qual é chamada a operação `intercala`) e a lista passada por parâmetro. Ambas listas originais devem ficar inalteradas e se elas tiverem tamanhos diferentes, os nós excedentes devem ser copiados para a nova lista. Como exemplo, a intercalação da lista implícita [3, 9, -4, 8] com a lista [-7, 2] deve resultar na nova lista [3, -7, 9, 2, -4, 8].

3. (25 Pontos) Considere o TAD `ListaDuplaCircular` para representar uma **lista circular duplamente encadeada** de inteiros como ilustrada na figura abaixo (ver os detalhes deste TAD no projeto).



   (a) Implemente a operação `void ListaDuplaCircular::remover(NoDuplo *p)`, que recebe como parâmetro o ponteiro `p` para um nó qualquer na lista que deve ser removido.

   (b) Implemente a operação `NoDuplo* ListaDuplaCircular::voltar(NoDuplo *p, int n)`, que recebe como parâmetro o ponteiro `p` para um nó qualquer na lista e retorna um ponteiro para o nó que esteja `n` posições antes de `p` na lista.

   (c) Implemente a operação `int ListaDuplaCircular::josephus(int k)` para o algoritmo de *Josephus*, cujo objetivo é remover seletivamente os nós da lista, saltando de `k` em `k` posições no sentido anti-horário, até que reste um único nó na lista. Iniciando pelo primeiro nó, o algoritmo deve retroceder `k` posições na lista e remover o nó daquela posição (utilize obrigatoriamente as operações `voltar` e `remover` para isto). Reiniciando a partir do nó à direita do nó removido anteriormente, o algoritmo deve novamente retroceder `k` posições na lista e remover o nó daquela posição. Este processo deve se repetir até que reste apenas um único nó na lista. Ao final, a operação deve retornar o valor armazenado no campo `info` deste nó.

4. (25 Pontos) Considere o TAD `MatrizEsparsa` para representar uma matriz de inteiros com uma grande quantidade de elementos nulos através de uma **lista contígua ordenada** contendo apenas os elementos diferentes de zero. O par de índices `(lin, col)` é utilizado como critério de ordenação dos elementos na lista. A operação `busca(int lin, int col)` retorna o valor -1 caso o elemento de posição `(lin, col)` esteja presente na lista. Caso o elemento não seja encontrado, ela retorna o índice da posição onde ele deveria estar na lista (ver os detalhes deste TAD no projeto).

   Implemente a operação `void MatrizEsparsa::insere(int lin, int col, int val)` para inserir um novo elemento não-nulo da matriz como um nó da lista. Caso já exista um elemento com os índices de linha e coluna `(lin, col)`, o algoritmo não deve inserir um novo nó. Utilize a operação `busca` para encontrar a posição adequada de inserção (ou atualização).