Nome: _____ Matrícula: _____

Leia atentamente as instruções abaixo:

- Fazer o download do arquivo `Avaliacao1EDLab2-2019-3.zip` do *site* do curso e descompactá-lo na sua máquina. Este arquivo contém todos os códigos para o desenvolvimento da prova.
- 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 prova. Caso sua solução tenha mais de uma função ou operação, elas devem estar entre esses marcadores.
- Colocar no arquivo `main.cpp` seu nome completo e número de matrícula.
- A prova é **individual** e **sem qualquer tipo de consulta**.
- Existe apenas um projeto do Code::Blocks que será usado na prova.
- Antes de sair do laboratório, enviar ao servidor – usando a janela de upload – cada arquivo de código que contém as respostas das questões da sua prova. Aguarde um momento e verá as suas respostas de cada questão da prova.
- **O desenvolvimento e envio do código são de inteira responsabilidade do aluno!**
- Endereço do servidor: `http://172.18.40.97:8080/edlab2ufjf/`

Questões:

1. (20 Pontos) Implemente uma função `troca` (defina adequadamente os parâmetros e retorno da função como solicitado no primeiro item que segue) e o corpo da função `void questao1()` no arquivo `main.cpp` com o que se pede:
   (a) Implemente a função `troca` de modo a ser possível trocar os valores de 2 variáveis inteiras ao se chamar essa função na `questao1`.
   (b) Crie duas variáveis inteiras `x`, com valor inicial 10, e `y`, com valor inicial 21.
   (c) Imprima os endereços de memória de `x` e `y`.
   (d) Use a função `troca` para alterar os valores entre `x` e `y`.
   (e) Declare um ponteiro para inteiro `p` e imprima o endereço de memória de `p`.
   (f) Faça com que `p` aponte para o endereço de memória de `x` e, usando o ponteiro `p`, imprima o endereço de memória e o valor de `x`.
   (g) Usando o ponteiro `p`, modifique o valor de `x` para 42.
   (h) Aloque dinamicamente um vetor com 2 inteiros, salve o ponteiro de retorno em `p` e inicialize o vetor com os valores [2, 4].
   (i) Use a função `troca` para alterar os valores entre os elementos de `p`.
   (j) Imprima os valores inteiros de `p` e libere a memória alocada dinamicamente.
2. (20 Pontos) Implemente a função `int strToInt(char *s, int n)` que converte **recursivamente** um vetor de caracteres para um número inteiro. Considere que o vetor possui um número válido. Lembre-se que é possível utilizar uma função auxiliar, caso julgue necessário.

```
int strToInt(char *s, int n);
```

3. (30 Pontos) Considere o TAD Caminho que representa uma sequência de índices de cidades visitadas sem repetição. A representação interna do caminho é definida por um vetor de inteiros, sendo que o valor `j` na posição `i` indica que `j` é a (i+1)-ésima cidade visitada. Por exemplo, se `vet[3]=2` então a 4ª cidade visitada é a cidade 2. Além disso, o TAD deve conter o número de cidades. Para este TAD, desenvolver:

   (a) Construtor e destrutor da classe. Para ter valores iniciais em `vet`, guarde o valor `i` em `vet[i]`.

   (b) Operação `int calculaDistanciaPercorrida(int **distancias)`, que recebe uma matriz de distancias e calcula a distância percorrida pelo caminho. Assuma que a matriz de distâncias é quadrada e que sua dimensão é igual ao número de cidades. A matriz contém as distâncias entre as cidades. Assim, a distância entre as cidades `i` e `j` é o valor guardado em `distancias[i][j]`. Portanto, a distância percorrida é a soma das distâncias entre as cidades na ordem definida pelo caminho.

   (c) Operação `void buscaGulosa(int **distancias)` que, a partir da matriz de distâncias, atualiza o vetor de cidades de modo que se tenha o menor caminho entre as cidades a partir da cidade de indice `0`. Deve-se manter a restrição de não repetir as cidades. Se julgar necessário, use a operação `void ordem(int **distancias, int i, int* ordem)`, que recebe a matriz de distâncias e o índice da cidade, e guarda no vetor `ordem` (que precisa ser alocado dinâmicamente) os índices das cidades em ordem de distância para a cidade `i`. Assim, o valor em `ordem[0]` é o índice da cidade mais próxima, `ordem[1]` é a segunda cidade mais próxima e assim por diante.

```
class Caminho {
public:
  Caminho(int n);
  ~Caminho();
  int calculaDistanciaPercorrida(int **distancias);
  void buscaGulosa(int **distancias);
  void imprimirCaminho();
private:
  void ordem(int **distancias, int i, int *ordem);
  int *vet;
  int n;
};
```

4. (30 Pontos) Considere matrizes quadradas e simétricas de ordem n em que os elementos da diagonal principal são iguais a zero. A matriz A que segue é um exemplo.

$$A = \begin{bmatrix} 0 & 4 & 7 & 3 & 6 \\ 4 & 0 & 1 & 7 & 9 \\ 7 & 1 & 0 & 9 & 7 \\ 3 & 7 & 9 & 0 & 4 \\ 6 & 9 & 7 & 4 & 0 \end{bmatrix}$$

O TAD `MatrizDistancias` representa esse tipo de matriz, que pode ser usado para guardar as distâncias entre cidades (como no exercício anterior). Os valores devem ser armazenados em `MatrizDistancias` numa representação linear com um único vetor (`vet`) e apenas os valores abaixo (ou acima) da diagonal principal devem ser guardados. Para esse TAD, desenvolver:

   (a) Construtor, que recebe a dimensão da matriz como parâmetro, e destrutor da classe.

   (b) Operação `int detInd(int i, int j)`, que recebe a linha `i` e a coluna `j` como parâmetros e retorna: o índice correspondente no vetor `vet`, -1 se o índice da linha ou da coluna forem inválidos, e -2 se `i` e `j` não representam um valor no vetor `vet`.

   (c) Operações `int get(int i, int j)` para acessar e `void set(int i, int j, int val)` para alterar valores da matriz. Se os índices `i` ou `j` forem inválidos, a operação `get` deve finalizar a execução do programa e a operação `set` deve imprimir uma mensagem de erro. Deve-se exibir uma mensagem de erro ao tentar atribuir um valor diferente de zero na diagonal principal.