UFJF/DCC Estrutura de Dados e Lab. de Programação II – TVC 1 23/04/2018

Nome: _____ Matrícula: _____

Leia atentamente as instruções abaixo:

- Fazer o download do arquivo `Avaliacao1EDLab2-2018-1.zip` do site do curso. Descompactá-lo em um diretório de 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, obrigatoriamente.
- 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) Dado um conjunto de valores inteiros positivos armazenados no vetor `dims` de tamanho `n`, construir uma função para criar e retornar um array bidimensional irregular (conhecido também como *jagged array*). Este array é representado por um vetor principal de tamanho `n` em que cada elemento aponta para um vetor secundário de tamanho `d`, sendo que `d` pode variar entre cada vetor secundário. Os valores de `d` são os valores passados como parâmetro no vetor de dimensões. Imprima o endereço de acesso a cada vetor secundário e retorne por referência o índice do maior vetor. Construa também uma função para destruir o vetor criado. Os protótipos são os que se seguem:

```
int** criaArrayIrregular(int* dims,int n,int *maxi);
void destroiArrayIrregular(int** vet, int n);

Exemplo de saída p/ 5 arrays de dimensões 6,7,5,9,1:
Vetor 0: endereco 0x397698
Vetor 1: endereco 0x3976b8
Vetor 2: endereco 0x3976e0
Vetor 3: endereco 0x397700
Vetor 4: endereco 0x396de0
Maior vetor: 3
```

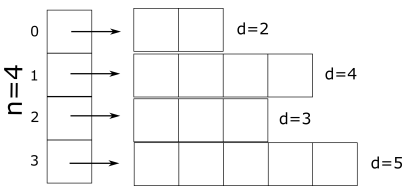

Ilustração de um array irregular

2. (20 Pontos) Desenvolver a função recursiva `somaDigitos()` para, dado um número inteiro positivo `val`, calcular e retornar a soma de seus dígitos. Além da soma, a função deve calcular o número de chamadas recursivas realizadas e guardar em `nr`, passado como referência. Por exemplo, se o número é 96541 a função deve calcular 25 e 5 para soma de dígitos e número de recursões, respectivamente.

```
int somaDigitos(int val, int *nr);
```

3. (30 Pontos) Abaixo encontram-se as classes que implementam os TADs `CopaDoMundo` e `Equipe`. Os dados armazenados para representar uma Copa do Mundo são o número de equipes e um vetor de equipes (ponteiro para TAD `Equipe`). O construtor do TAD `Equipe` inicializa os atributos com valores aleatórios. Para o TAD `CopaDoMundo`, desenvolver:

   (a) construtor (que recebe o número de equipes da copa) e destrutor da classe;

   (b) operação `void imprime()` que imprime as informações de todas as equipes (grupo e pontos), além das equipes que passaram de fase (com pontuação acima da média);

   (c) operação `void primeiraGrupo(int *g, int *p)` que retorna (através de ponteiros) o grupo de menor número e a pontuação da equipe que mais pontuou nesse grupo.

```
class CopaDoMundo {
  private:
    int n;
    Equipe *equipes;
  public:
    CopaDoMundo(int tam);
    ~CopaDoMundo();
    void imprime();
    void primeiraGrupo(int *g, int *p);
};
```

```
class Equipe {
  private:
    int grupo, pontos;
  public:
    Equipe(); ~Equipe();
    int getGrupo();
    void setGrupo(int g);
    int getPontos();
    void setPontos(int p);
};
```

4. (30 Pontos) Considere uma matriz quadrada em blocos, isto é, onde seus elementos são submatrizes, como nos exemplos abaixos:

$$A = \begin{bmatrix} A_{00} & A_{01} & \dots & A_{0n} \\ A_{10} & A_{11} & \dots & A_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{20} & A_{21} & \dots & A_{nn} \end{bmatrix}, \qquad A_D = \begin{bmatrix} A_{00} & 0 & 0 \\ 0 & A_{11} & 0 \\ 0 & 0 & A_{22}, \end{bmatrix}$$

onde A_{ij} são matrizes quadradas $t_b \times t_b$. Por exemplo, se $t_b = 2$, a matriz A teria dimensão $2n \times 2n$. Um caso particular de matrizes em bloco são as matrizes em bloco diagonal que assumem a forma da matriz A_D acima, onde os 0 representam matrizes $n_b \times n$ com valores iguais a zero. Note que nos exemplos acima $n_b = 3$, mas esse número poderia ser qualquer.

Desenvolver um TAD para representar uma matriz bloco diagonal cujo tamanho do bloco é fixado em 3 através da representação linear para cada bloco. Os blocos devem ser armazenados sequencialmente no vetor e para cada bloco utilize a representação linear. O número de blocos da matriz deve ser um parâmetro do construtor. Para esse TAD, desenvolva:

   (a) construtor e destrutor da classe;

   (b) operação `int detInd(int i, int j)` que retorna -1 se os indices `i` ou `j` são inválidos; -2 se o acesso é fora dos blocos (onde o valor é zero); ou o índice k para acessar a posição do vetor linear. Dica: *calcule inicialmente o índice do bloco `ib` e `jb` e verifique se o acesso é em um bloco da diagonal ou não;*

   (c) as operações `float get(int i, int j)` e `void set(int i, int j, float val)` que retorna e altera o valor na posição `i,j` da matriz, respectivamente. Usar a função `detInd()`;

   (d) desenvolver uma operação para atribuir `val` a todos elementos de um determinado bloco (`setBloco(int ib, float val)`, onde `ib` é o índice do bloco.

```
class MatrizBlocoDiagonal {
public:
  MatrizBloco(int numeroBlocos);
  ~MatrizBloco();
  float get(int i, int j);
  void set(int i, int j, float val);
  void setBloco(int ib, float val);
```

```
// continua ...
private:
  int n;       // dimensao da matriz
  int numBlocos; // numero de blocos
  float *vet;
  int detInd(int i, int j);
};
```