Nome: _____ Matrícula: _____

1. (20 Pontos) Implemente a função `int* vetorNegativos(int vet[], int n, int *t)`, que recebe como parâmetros um vetor de números inteiros `vet` e seu tamanho `n`, e retorna um novo vetor com os elementos negativos de `vet`. O novo vetor deve ser alocado com o número adequado de posições. Se não houver nenhum valor negativo no vetor, a função deve retornar `NULL`. O tamanho do novo vetor deve ser armazenado em `t`.

```
int* vetorNegativos(int vet[], int n, int *t);
```

```
int* vetorNegativos(int vet[], int n, int *t)
{
    *t = 0;
    for(int i = 0; i < n; i++)
        if(vet[i] < 0)
            (*t)++;
    if(*t == 0)
        return NULL;
    int *r = new int[*t];
    int j = 0;
    for(int i = 0; i < n; i++)
        if(vet[i] < 0)
            r[j++] = vet[i];
    return r;
}
```

10 — Declaração, retorno e cálculo da quantidade de elementos negativos (em *t).

10 — Declaração, alocação e retorno do vetor contendo a apenas os elementos negativos (em v[ ]).

2. (20 Pontos) Implemente a função `bool verificaIguais(int vet1[], int vet2[], int n)`, que recebe como parâmetros dois vetores de inteiros `vet1` e `vet2` de mesmo tamanho `n`. A função deve verificar **recursivamente** se os dois vetores são iguais (possuem os mesmos valores). Caso não sejam iguais, a função deve retornar `false`. Caso os vetores sejam iguais, a função deve retornar `true`.

```
bool verificaIguais(int vet1[], int vet2[], int n);
```

Solução 1

```
bool verificaIguais(int vet1[], int vet2[], int n)
{
    if(n == 0)
        return true;
    if(vet1[n-1] != vet2[n-1])
        return false;
    return verificaIguais(vet1, vet2, n-1);
}
```

5 — Caso base 1 - vetores com 0 elementos são iguais

7 — Caso base 2 - elementos da posição n-1 diferentes

8 — Caso geral. Diminui o tamanho dos vetores de 1

Solução 2

```
bool verificaIguais2(int vet1[], int vet2[], int n)
{
    if(n == 0)
        return true;
    if(vet1[0] != vet2[0])
        return false;
    return verificaIguais2(vet1+1, vet2+1, n-1);
}
```

5 — Caso base 1 - vetores com 0 elementos são iguais

7 — Caso base 2 - elementos da posição 0 diferentes

8 — Caso geral. Avança os ponteiros para a próxima posição (faz com que o elemento da posição 0 altera em cada chamada)

3. (30 Pontos) A seguir encontra-se a classe que implementa o TAD `Avaliacao`. Os dados armazenados para representar uma avaliação são:

   - `int questoes` (número inteiro positivo indicando o total de questões de uma atividade);
   - `float valor` (valor das questões);
   - `float *notas` (vetor contendo as notas obtidas em cada questão).

   Para o TAD `Avaliacao`, desenvolver:

   (a) construtor, que recebe o total de questões (assuma que todas as questões possuem o mesmo valor e que o valor total da atividade é 100);

   (b) destrutor;

   (c) operação `void leNotas()`, que solicita ao usuário que digite a nota de cada questão (notas fora do intervalo válido devem ser lidas novamente);

   (d) operação `void relatorio()`, que calcula e imprime a nota final, além de imprimir mensagens indicando em quais questões as notas ficaram abaixo de 60%.

```
class Avaliacao
{
  private:
    int questoes;
    float valor;
    float *notas;

  public:
    Avaliacao(int n);
    ~Avaliacao();
    void leNotas();
    void relatorio();
};
```

```
Avaliacao::Avaliacao(int n)
{
    if(n <= 0)
    {
        cout << "A atividade deve ter pelo menos 1 questao";
        exit(1);
    }
    questoes = n;
    notas = new float[questoes];
    valor = 100.0 / questoes;
}
```

9

```
Avaliacao::~Avaliacao()
{
    delete [] notas;
}
```

3

9

```
void Avaliacao::leNotas()
{
    for(int i = 0; i < questoes; i++)
    {
        cout << "Digite a nota da questao " << i+1 << ": ";
        cin >> notas[i];
        while(notas[i] < 0 || notas[i] > valor)
        {
            cout << "Nota invalida! Digite novamente: ";
            cin >> notas[i];
        }
    }
}
```

9

```
void Avaliacao::relatorio()
{
    float notaFinal = 0;
    for(int i = 0; i < questoes; i++)
    {
        notaFinal += notas[i];
        if(notas[i] < valor * 0.6)
            cout << "Nota da questao " << i+1 << " abaixo de 60%" << endl;
    }
    cout << "Nota final: " << notaFinal << endl;
}
```

4. (30 Pontos) Considere matrizes quadradas e simétricas de ordem n em que os elementos acima da diagonal principal são iguais à diferença entre os índices da respectiva coluna e da respectiva linha. Já os elementos abaixo da diagonal principal são iguais à diferença entre os índices da respectiva linha e da respectiva coluna. A matriz A é um exemplo desse tipo de matriz.

$$A = \begin{bmatrix} -5 & 1 & 2 & 3 & 4 \\ 1 & 6 & 1 & 2 & 3 \\ 2 & 1 & -8 & 1 & 2 \\ 3 & 2 & 1 & 7 & 1 \\ 4 & 3 & 2 & 1 & -9 \end{bmatrix}$$

O TAD `MatrizEspecial` representa esse tipo de matriz. Os valores devem ser armazenados no TAD `MatrizEspecial` numa representação linear com um único vetor (`vet`) e de modo que a quantidade de elementos armazenados seja mínima. Para esse TAD, desenvolver:

(a) Construtor, que recebe a ordem 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 inválido para os elementos fora da diagonal principal.

5

```
MatrizEspecial::MatrizEspecial(int nn)
{
    n = nn;
    vet = new int[n];
}
```

3

```
MatrizEspecial::~MatrizEspecial()
{
    delete [] vet;
}
```

6

```
int MatrizEspecial::detInd(int i, int j)
{
    if(i >= 0 && i < n && j >= 0 && j < n)
    {
        if(i == j)
            return i;
        return -2;
    }
    return -1;
}
```

8

```
int MatrizEspecial::get(int i, int j)
{
    int k = detInd(i, j);
    if(k == -1)
    {
        cout << "indices invalidos" << endl;
        exit(1);
    }
    else if(k == -2)
    {
        if(i > j)
            return i-j;
        return j-i;
    }
    else
        return vet[k];
}
```

8

```
void MatrizEspecial::set(int i, int j, int val)
{
    int k = detInd(i, j);
    if(k == -1)
        cout << "indices invalidos" << endl;
    else if(k == -2)
    {
        if(i > j && val != i-j)
            cout << "Tentando atribuir valor invalido" << endl;
        else if(val != j-i)
            cout << "Tentando atribuir valor invalido" << endl;
    }
    else
        vet[i] = val;
}
```