UNIVALI Ciência da Computação – CTTMar Kobrasol PROVA P3 Data: 05/12/2016
Disciplina: ALGORITMOS E PROGRAMAÇÃO – 1ºper. **Profa:** Fernanda dos S. Cunha

1. (**2,0 pontos – resolver em pseudocódigo VisuAlg ou Portugol)** Um número é dito ser capicua quando lido da esquerda para a direita é o mesmo que quando lido da direita para a esquerda. O ano 2002, por exemplo, é capicua. Então, elabore uma função que receba como parâmetro um número inteiro e verifique matematicamente se um número possui essa característica. Considere para teste apenas números com 4 dígitos. Caso o número seja capicua, a função deve retornar verdadeiro, e falso em caso contrário.

```
FUNCAO ehCapicua (num: INTEIRO) : LOGICO          // existem outras versoes
VAR
    num2: INTEIRO
INICIO
    num2 <- num DIV 1000                  // pega milhar do num e atribui ao num2
    num <- num MOD 1000                   // posso mudar pq não é a var original, é a cópia
    num2 <- num2 + (num DIV 100) * 10     // pega centena do num, multiplica por 10 e soma
    num <- num MOD 100
    num2 <- num2 + (num DIV 10) * 100     // pega decimal do num, multiplica por 100 e soma
    num <- num MOD 10
    num2 <- num2 + num * 1000             // pega unidade do num, multiplica por 1000 e soma
    SE (num2 = num) ENTAO
        retorne  VERDADEIRO
    SENAO
        retorne  FALSO
    FIMSE
FIMFUNCAO
```

2. (**3,0 pontos – resolver em linguagem C++)** Construa uma função sem retorno, que receba três coeficientes (inteiros) relativos à uma equação de 2º grau ($ax^2 + bx + c = 0$), calcule e devolva suas raízes através da fórmula de Báscara:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \qquad \Delta = b^2 - 4ac$$

Deve-se considerar a possibilidade da existência de nenhuma (Δ<0), uma (Δ=0) ou duas raízes (Δ>0). Apresente também como seria uma chamada a esta função em um programa qualquer.

```
void raizesEq2oGrau ( int a, int b, int c, int &numRaizes, float &x1, float &x2 ) {
    float delta = b*b – 4*a*c;
    if ( delta<0 ) {
        numRaizes = 0; // x1 e x2 não existem, nem precisam receber valor algum pq não são válidos
    }
    else {
        if ( delta==0 ) {
            numRaizes = 1;  x1 = - b / (2*a);        // x2 não existe, logo não recebe nenhum valor
        }
        else {
            numRaizes = 2;
            x1 = (- b + pow(delta, 0.5)) / (2*a);
            x2 = (- b - pow(delta, 0.5)) / (2*a);
        }
    }
}
```

Testes: nenhuma raiz para $x^2 - 4x + 5 = 0$ / uma raiz para $4x^2 - 4x + 1 = 0$ / duas raizes para $x^2 - 5x + 6 = 0$
Para a equação $x^2 - 5x + 6$ a chamada é: raizesEq2oGrau (1, -5, 6, *nraizes, r1, r2*);
onde os últimos parâmetros serão as respostas e trarão respectivamente 2, 3, 2.

3. (**5,0 pontos - total)** A distância (em km) entre cinco cidades é dada pela tabela abaixo:

	1	2	3	4	5
1	-	15	30	5	12
2	15	-	10	17	28
3	30	10	-	3	11
4	5	17	3	-	80
5	12	28	11	80	-

Tomando esta matriz como exemplo, escreva um **algoritmo ou programa modularizado** para armazenar as distâncias entre **n** (3 ≤ n ≤ 10) cidades numa matriz, mostrar a distância entre vários pares de cidades informados pelo usuário, e por fim, calcular e apresentar a distância a ser percorrida para um determinado percurso entre as cidades.

Para tanto, além do programa principal/main (1,0) desenvolva:

- (1,5) procedimento que inicialize a matriz com as distâncias entre as cidades, de forma otimizada;
- (1,0) função para buscar a distância entre um par de cidades passados por parâmetro;
- (1,5) função para calcular a distância percorrida para um determinado percurso (sem tamanho determinado), reutilizando de alguma maneira a função anterior.
  Ex.: dado o percurso 1, 2, 3, 2, 5, 4 => a distância percorrida é 15 + 10 + 10 + 28 + 80 = 143 km.
  Dica: não usar vetor para resolver este item.

Considere conhecida a função leiaNumeroInteiroIntervalo(inf, sup: INTEIRO): INTEIRO que poderá ser usada caso necessário. Em C++: int leiaNumeroInteiroIntervalo(int inf, int sup).

```
void montaMatrizSimetricaOtimizada(int ncid, int matD[ ][10]) {
   for(int i=0; i < ncid-1; i++)  {
      matD[i][i] = 0;          // diagonal zerada
      for(int j = i+1; j < ncid; j++) {
         cout << "Distancia entre cidades "<< i+1 << " e " << j+1 <<": ";
         matD[ i ][ j ] = leiaNumeroInteiroIntervalo(1,32000);
         matD[ j ][ i ] = matD[ i ][ j ];       // copia para parte inferior da matriz
      }
   }
   matD[ncid-1][ncid-1]=0;          // ultima posicao da matriz/diagonal, onde o for nao passa
}
int pegaDistanciaParCidades(int cid1, int cid2, int matD[ ][10]) {
   return matD[ cid1-1 ][ cid2-1 ];
}
int pegaDistanciaPercurso(int ncid, int matD[ ][10]) {
   int cid1, cid2, soma=0;          char resp;
   cout << "Entre com a cidade inicial do percurso: ";     cid1 = leiaNumeroInteiroIntervalo(1,ncid);
   do  {
      cout << "Entre com a próxima cidade: ";              cid2 = leiaNumeroInteiroIntervalo(1,ncid);
      if( cid1 != cid2) {
          soma += pegaDistanciaParCidades(cid1, cid2, matD);
          cid1= cid2;
      }
      cin.ignore();
      do {
         cout << "Continuar? S/N: ";         resp = toupper(cin.get());
      } while(resp!='S' && resp!='N');
   } while(resp=='S');
   return soma;
}
```

```
int main() {
   int cid1, cid2, ncid, mat[10][10], result;
   char resp;
   cout << "Entre com n. de cidades: ";
   ncid = leiaNumeroInteiroIntervalo(3,10);
   montaMatrizSimetricaOtimizada(ncid, mat);
   do {
      cout << "Par de cidades\n – Cidade 1: ";     cid1 = leiaNumeroInteiroIntervalo(1,ncid);
      cout << " – Cidade 2: ";                     cid2 = leiaNumeroInteiroIntervalo(1,ncid);
      cout << "Distancia direta: " << pegaDistanciaParCidades(cid1, cid2, mat) << endl;
      cin.ignore();
      do {
         cout << "Continuar? S/N: ";          resp = toupper(cin.get());
      } while (resp!='S' && resp!='N');
   } while(resp=='S');
   result = pegaDistanciaPercurso(ncid, mat);
   cout << "Distancia total = " << result;
   return 1;
}
```