1.

```
#include <stdio.h>
#include <stdlib.h>

#define N 4

/* Geramos quadrados de lado progressivamente menor. A cada iteração, o lado é
   reduzido em 2 e o canto é aumentado em 1, resultando em um quadrado mais
   interno. */
void geraMatrizInca (int matriz [N][N])
{
    int i, j;
    int valor_atual = 1;
    int canto_atual = 0;
    int lado = N;

    while (lado > 0) {
        /* Vai para a direita. */
        for (j = canto_atual; j < canto_atual + lado; j++)
            matriz [canto_atual][j] = valor_atual++;

        /* Vai para baixo. Pula a posição mais acima. */
        for (i = canto_atual+1; i < canto_atual + lado; i++)
            matriz [i][canto_atual+lado-1] = valor_atual++;

        /* Vai para a esquerda. Pula a posição mais à direita. */
        for (j = canto_atual+lado-2; j >= canto_atual; j--)
            matriz [canto_atual+lado-1][j] = valor_atual++;

        /* Vai para cima. Pula as posições mais abaixo e mais acima. */
        for (i = canto_atual+lado-2; i > canto_atual; i--)
            matriz [i][canto_atual] = valor_atual++;

        lado -= 2;
        canto_atual++;
    }
}

int main ()
{
    int i, j;
    int matriz [N][N];

    /* Gera */
    geraMatrizInca (matriz);

    /* Mostra */
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            printf ("%d\t", matriz [i][j]);
        printf ("\n");
    }
    return (0);
}
```

2.

```
#include <stdio.h>
#include <stdlib.h>

#define N_LINHAS 5
#define N_CARACTERES 5

/* Versão 1: armazenando a saída toda em uma matriz. Note que aqui eu estou
   usando as matrizes como vetores de strings. */
int main ()
{
    int i, j;

    /* Declaramos o padrão e a saída com 1 caractere a mais por linha para poder
       usar a notação de strings - a posição extra é para o '\0'. */
    char padrao [N_LINHAS][N_CARACTERES+1] =
        {{"X->->"}, {"|X* +"}, {"V X| "}, {"|  Xa"}, {"v  -X"}};
    char saida [N_LINHAS*2][N_CARACTERES*2+1];

    /* Geramos a saída. Cada caractere do padrão é colocado em 4 posições da
       saída, sempre mantendo a mesma distância de um dos 4 cantos (superior
       esquerdo, superior direito, inferior esquerdo e inferior direito). */
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_CARACTERES; j++)
        {
            saida [i][j] = padrao [i][j];
            saida [i][N_CARACTERES*2-1-j] = padrao [i][j];
            saida [N_LINHAS*2-1-i][j] = padrao [i][j];
            saida [N_LINHAS*2-1-i][N_CARACTERES*2-1-j] = padrao [i][j];
        }
    }

    for (i = 0; i < N_LINHAS*2; i++)
         saida [i][N_CARACTERES*2] = '\0'; /* Fecha a linha. */

    /* Mostra a saída. */
    for (i = 0; i < N_LINHAS*2; i++)
        printf ("%s\n", saida [i]);

    return (0);
}
```

```
/* Versão 2: jogando o padrão direto na saída. Aqui, fica mais fácil imprimir
   caractere por caractere. */

int main ()
{
    int i, j;
    char padrao [N_LINHAS][N_CARACTERES] =
        {{'X', '-', '>', '-', '>'},
         {'|', 'X', '*', ' ', '+'},
         {'V', ' ', 'X', '|', ' '},
         {'|', ' ', ' ', 'X', 'a'},
         {'v', ' ', ' ', '-', 'X'}};

    /* Primeiro, percorremos as linhas na ordem normal, e imprimimos cada linha
       2 vezes: na ordem normal e inversa. */
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_CARACTERES; j++)
            printf ("%c", padrao [i][j]);
        for (j = N_CARACTERES-1; j >= 0; j--)
            printf ("%c", padrao [i][j]);
        printf ("\n");
    }

    /* Agora, fazemos a mesma coisa, mas com as linhas em ordem inversa. */
    for (i = N_LINHAS-1; i >= 0; i--)
    {
        for (j = 0; j < N_CARACTERES; j++)
            printf ("%c", padrao [i][j]);
        for (j = N_CARACTERES-1; j >= 0; j--)
            printf ("%c", padrao [i][j]);
        printf ("\n");
    }

    return (0);
}
```

3.

```
#define N_AEROPORTOS 5
#define N_VOOS 5

/*--------------------------------------------------------------------------*/
// Funçõezinhas auxiliares. Seria melhor fazer sem tamanho fixo, mas para isso,
// precisaríamos usar alocação dinâmica...

void mostraMatriz (int m [N_AEROPORTOS][N_AEROPORTOS])
{
    int i, j;

    for (i = 0; i < N_AEROPORTOS; i++)
    {
        for (j = 0; j < N_AEROPORTOS; j++)
            printf ("%d ", m [i][j]);
        printf ("\n");
    }
    printf ("\n");
}

/*--------------------------------------------------------------------------*/

void copiaMatriz (int orig [N_AEROPORTOS][N_AEROPORTOS],
                  int dest [N_AEROPORTOS][N_AEROPORTOS])
{
    int i, j;

    for (i = 0; i < N_AEROPORTOS; i++)
        for (j = 0; j < N_AEROPORTOS; j++)
            dest [i][j] = orig [i][j];
}

/*--------------------------------------------------------------------------*/

void multiplicaMatrizes (int m1 [N_AEROPORTOS][N_AEROPORTOS],
                         int m2 [N_AEROPORTOS][N_AEROPORTOS],
                         int out [N_AEROPORTOS][N_AEROPORTOS])
{
    int i, j, k;

    // Para cada posição [i][j] da saída...
    for (i = 0; i < N_AEROPORTOS; i++)
    {
        for (j = 0; j < N_AEROPORTOS; j++)
        {
            out [i][j] = 0;
            // ... percorre a linha i de m1 * coluna j da m2.
            for (k = 0; k < N_AEROPORTOS; k++)
                out [i][j] += m1 [i][k] * m2 [k][j];
        }
    }
}

/*--------------------------------------------------------------------------*/
// 1. Preenche a matriz de saída para ser a matriz de adjacência.

void geraMatrizAdjacencia (int voos [N_VOOS][2], int out [N_AEROPORTOS][N_AEROPORTOS])
{
    int i, j, orig, dest;

    // Como o conteúdo inicial da out é indeterminado, vamos começar zerando ela.
    for (i = 0; i < N_AEROPORTOS; i++)
        for (j = 0; j < N_AEROPORTOS; j++)
            out [i][j] = 0;

    // Agora, cada voo atualiza DUAS posições da saída.
    for (i = 0; i < N_VOOS; i++)
    {
        orig = voos [i][0];
```

```
        dest = voos [i][1];
        out [orig][dest] = 1;
        out [dest][orig] = 1;
    }
}

/*-------------------------------------------------------------------------*/
// 2. Identifica o aeroporto mais movimentado.

int aeroportoMaisMovimentado (int m [N_AEROPORTOS][N_AEROPORTOS])
{
    int i, j, soma, maior_soma, mais_movimentado;

    // Supondo que nenhum voo parte e chega no mesmo lugar, é só ver a linha
    // com a maior soma. Não precisa nem multiplicar por 2 e nem ver na
    // vertical para considerar idas e vindas. Você consegue ver o motivo?
    maior_soma = 0;

    for (i = 0; i < N_AEROPORTOS; i++) // Para cada aeroporto...
    {
        // ... conta quantos voos passam por ele.
        soma = 0;
        for (j = 0; j < N_AEROPORTOS; j++)
            soma += m [i][j];

        if (soma > maior_soma)
        {
            maior_soma = soma;
            mais_movimentado = i;
        }
    }

    return (mais_movimentado);
}

/*-------------------------------------------------------------------------*/
// 3. Função super-específica, conta os trajetos entre 2 aeroportos com
// exatamente e com até X conexões. O truque é multiplicar a matriz de
// adjacência por si mesma sucessivamente - após X-1 multiplicações, o número
// na posição i,j da matriz resultante é o número de trajetos com EXATAMENTE
// X conexões. Para saber o número de trajetos com ATÉ X conexões, basta
// acumular os valoes de todas as multiplicações sucessivas.

void descobreTrajetosComHops (int madj [N_AEROPORTOS][N_AEROPORTOS], int hops, int
com_n_hops [N_AEROPORTOS][N_AEROPORTOS], int com_ate_n_hops [N_AEROPORTOS][N_AEROPORTOS])
{
    int hop, i, j;

    int matmul_aux [N_AEROPORTOS][N_AEROPORTOS]; // Para o resultado da multiplicação.

    // Começa com 1 hop (salto), ou seja, a própria matriz de adjacência.
    copiaMatriz (madj, com_n_hops);
    copiaMatriz (madj, com_ate_n_hops);

    // Agora vai aumentando até atingir o número de hops.
    for (hop = 1; hop < hops; hop++)
    {
        multiplicaMatrizes (com_n_hops, madj, matmul_aux);
        copiaMatriz (matmul_aux, com_n_hops);

        // Acumula.
        for (i = 0; i < N_AEROPORTOS; i++)
            for (j = 0; j < N_AEROPORTOS; j++)
                com_ate_n_hops [i][j] += com_n_hops [i][j];
    }
}
```

```
/*-------------------------------------------------------------------------*/
/* Main para a 1 e 2. */

int main ()
{
    int mais_movimentado;
    int voos [N_VOOS][2] = {{0,3},{1,3},{2,3},{4,3},{4,5}};
    int madjacencia [N_AEROPORTOS][N_AEROPORTOS];

    geraMatrizAdjacencia (voos, madjacencia);

    mostraMatriz (madjacencia);

    mais_movimentado = aeroportoMaisMovimentado (madjacencia);
    printf ("O mais movimentado eh o %d.\n", mais_movimentado);

    return (0);
}

/*-------------------------------------------------------------------------*/
/* Main para a 3. */

int main ()
{
    int voos [N_VOOS][2] = {{0,3},{1,3},{2,3},{4,3},{4,5}};
    int madjacencia [N_AEROPORTOS][N_AEROPORTOS];
    int com_n_hops [N_AEROPORTOS][N_AEROPORTOS];
    int com_ate_n_hops [N_AEROPORTOS][N_AEROPORTOS];
    int orig, dest, hops;

    geraMatrizAdjacencia (voos, madjacencia);

    printf ("Digite o numero da origiem, do destino, e de conexoes.\n");
    scanf ("%d %d %d", &orig, &dest, &hops); // Supondo que está tudo ok.

    // a: este é simples, a matriz de adjacência já diz o que precisamos.
    if (madjacencia [orig][dest])
        printf ("\nExiste conexao direta entre %d e %d.\n", orig, dest);
    else
        printf ("\nSem conexao direta entre %d e %d.\n", orig, dest);

    // b e c: vou fazer em uma função!
    descobreTrajetosComHops (madjacencia, hops, com_n_hops, com_ate_n_hops);

    // Só para visualizar.
    printf ("\nMatriz de adjacencia:\n");
    mostraMatriz (madjacencia);

    printf ("Trajetos com %d conexoes:\n", hops);
    mostraMatriz (com_n_hops);

    printf ("Trajetos com ate %d conexoes:\n", hops);
    mostraMatriz (com_ate_n_hops);

    printf ("Trajetos entre %d e %d com exatamente %d conexoes: %d\n",
             orig, dest, hops, com_n_hops [orig][dest]);

    printf ("Trajetos entre %d e %d com ate %d conexoes: %d\n",
             orig, dest, hops, com_ate_n_hops [orig][dest]);

    return 0 ;
}
```