1.

```
#include <stdio.h>

#define N_LINHAS 3
#define N_COLUNAS 4

int main ()
{
    int i, j, soma, melhor_soma;
    int campo [N_LINHAS][N_COLUNAS] = {{81, 28, 240, 10},
                                       {40, 10, 100, 240},
                                       {20, 180, 110, 35}};

    melhor_soma = 0;

    // Testa cada LINHA.
    for (i = 0; i < N_LINHAS; i++)
    {
        soma = 0;

        // Percorre todas as colunas desta linha.
        for (j = 0; j < N_COLUNAS; j++)
            soma += campo [i][j];

        if (soma > melhor_soma)
            melhor_soma = soma;
    }

    // Quase igual, mas agora testa cada COLUNA.
    for (j = 0; j < N_COLUNAS; j++)
    {
        soma = 0;

        // Percorre todas as linhas desta coluna.
        for (i = 0; i < N_LINHAS; i++)
            soma += campo [i][j];

        if (soma > melhor_soma)
            melhor_soma = soma;
    }


    // Mostra o maior total em alguma linha ou coluna.
    printf ("%d\n", melhor_soma);

    return 0;
}
```

```c
/* Solução alternativa: a versão mais simples tem 2 blocos muito parecidos, para
 percorrer em linhas e em colunas. Uma forma de percorrer a matriz uma única vez
 é guardar as somas para todas as linhas e colunas independentemente. Esta
 solução é um pouco mais "limpa", mas usa mais memória. Daria também para fazer
 uma mistura das duas soluções, testando diretamente a soma de cada linha mas
 guardando as somas das colunas, mas seria uma solução bem deselegante. */

#include <stdio.h>

#define N_LINHAS 3
#define N_COLUNAS 4

int main ()
{
    int i, j, melhor_soma;
    int campo [N_LINHAS][N_COLUNAS] = {{81, 28, 240, 10},
                                       {40, 10, 100, 240},
                                       {20, 180, 110, 35}};
    int somas_linha [N_LINHAS];
    int somas_coluna [N_COLUNAS];

    // Inicializa todas as somas em 0.
    for (i = 0; i < N_LINHAS; i++)
        somas_linha [i] = 0;
    for (i = 0; i < N_COLUNAS; i++)
        somas_coluna [i] = 0;

    // Percorre o campo e atualiza para cada célula a linha e a coluna
    // correspondente.
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_COLUNAS; j++)
        {
            somas_linha [i] += campo [i][j];
            somas_coluna [j] += campo [i][j];
        }
    }

    // Procura o maior valor.
    melhor_soma = 0;
    for (i = 0; i < N_LINHAS; i++)
        if (somas_linha [i] > melhor_soma)
            melhor_soma = somas_linha [i];
    for (i = 0; i < N_COLUNAS; i++)
        if (somas_coluna [i] > melhor_soma)
            melhor_soma = somas_coluna [i];

    // Mostra o maior total em alguma linha ou coluna.
    printf ("%d\n", melhor_soma);

    return 0;
}
```

2.

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

#define N_STRINGS 5
#define BUFLEN 128

int main ()
{
    int i;
    char vetor_de_strings [N_STRINGS][BUFLEN];

    for (i = 0; i < N_STRINGS; i++)
        fgets (vetor_de_strings [i], BUFLEN, stdin);

    for (i = N_STRINGS-1; i >= 0; i--)
        printf ("%s", vetor_de_strings [i]);

    return (0);
}
```

3.

```
#include <stdio.h>

#define N 6

int main ()
{
    int i, j;
    int tabuleiro [N][N] = {{0,0,1,0,0,0},
                            {1,9,9,9,9,9},
                            {0,9,9,9,9,9},
                            {0,9,9,9,9,9},
                            {1,9,9,9,9,9},
                            {1,9,9,9,9,9}};

    /* A resposta é mais simples do que parece. Como branco = 1, você pode somar
    os valores nas 3 posições vizinhas e ver se a soma é maior ou igual a 2. */
    for (i = 1; i < N; i++)
        for (j = 1; j < N; j++)
            if (tabuleiro [i-1][j] + tabuleiro [i][j-1] +
                tabuleiro [i-1][j-1] >= 2)
                tabuleiro [i][j] = 0;
            else
                tabuleiro [i][j] = 1;

    // Mostra.
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            printf ("%d", tabuleiro [i][j]);
        printf ("\n");
    }

    return 0;
}
```

4.

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

#define M 2
#define N 3

int main ()
{
    int i, j, k, total;
    int matriz1 [M][N];
    int matriz2 [N][M];

    /* Preenche e mostra. */
    srand (time (NULL));
    printf ("Matriz 1:\n");
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            matriz1 [i][j] = rand () % 10;
            printf ("%d ", matriz1 [i][j]);
        }
        printf ("\n");
    }
    printf ("\n");

    printf ("Matriz 2:\n");
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            matriz2 [i][j] = rand () % 10;
            printf ("%d ", matriz2 [i][j]);
        }
        printf ("\n");
    }
    printf ("\n");

    /* Multiplica. Aqui, o desafio é enxergar como as linhas e colunas precisam
      ser percorridas. Cada uma das M linhas da matriz1 é percorrida M vezes.
      Uma linha é percorrida em um loop de N iterações. Isso quer dizer que
      temos não dois, mas TRÊS loops aninhados! */
    for (i = 0; i < M; i++) /* Cada linha da matriz 1... */
    {
        for (j = 0; j < M; j++) /* ... é percorrida M vezes. */
        {
            total = 0;
            for (k = 0; k < N; k++)
                /* Linha i da matriz1 x coluna j da matriz2. */
                total += matriz1 [i][k] * matriz2 [k][j];
            printf ("%d ", total);
        }
        printf ("\n");
    }

    return (0);
}
```

5.

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

/* Função auxiliar que zera um vetor. */
void zeraVetor (int* vetor, int n) {
    int i;
    for (i = 0; i < n; i++)
        vetor [i] = 0;
}

/* Para resolver este exercício, usaremos um vetor de 10 posições que diz quais
  números já apareceram na solução em uma determinada linha/coluna/bloco. */

int main ()
{
    int i, j, block_row, block_col;
    int usados [10]; /* Vetor que diz quais números já foram usados. */
    int solucao_valida = 1; /* Começamos supondo que a solução é válida. */
    int resposta [9][9] =
        {{9, 5, 4, 8, 1, 6, 3, 7, 2},
         {7, 8, 6, 2, 5, 3, 1, 4, 9},
         {1, 2, 3, 7, 9, 4, 6, 5, 8},
         {3, 1, 8, 9, 7, 2, 4, 6, 5},
         {2, 7, 9, 4, 6, 5, 8, 1, 3},
         {4, 6, 5, 3, 8, 1, 9, 2, 7},
         {8, 4, 7, 1, 2, 9, 5, 3, 6},
         {5, 3, 2, 6, 4, 8, 7, 9, 1},
         {6, 9, 1, 5, 3, 7, 2, 8, 4}};

    /* Tudo entre 1 e 9? */
    for (i = 0; i < 9 && solucao_valida; i++)
        for (j = 0; j < 9 && solucao_valida; j++)
            if (resposta [i][j] < 1 || resposta [i][j] > 9)
                solucao_valida = 0; /* Número inválido. */

    /* Para cada linha... */
    for (i = 0; i < 9 && solucao_valida; i++) {
        zeraVetor (usados, 10); /* Zera o vetor de números usados. */
        for (j = 0; j < 9 && solucao_valida; j++)
            if (usados [resposta [i][j]] != 0)
                solucao_valida = 0; /* Este número já apareceu. */
            else
                usados [resposta [i][j]] = 1;
    }

    /* Agora em colunas. */
    for (j = 0; j < 9 && solucao_valida; j++) {
        zeraVetor (usados, 10); /* Zera o vetor de números usados. */
        for (i = 0; i < 9 && solucao_valida; i++)
            if (usados [resposta [i][j]] != 0)
                solucao_valida = 0; /* Este número já apareceu. */
            else
                usados [resposta [i][j]] = 1;
    }

    /* Agora em blocos. */
    for (block_row = 0; block_row < 3 && solucao_valida; block_row++)
        for (block_col = 0; block_col < 3 && solucao_valida; block_col++) {
            zeraVetor (usados, 10); /* Zera o vetor de números usados.*/
            /* Percorre este bloco apenas. */
            for (i = 0; i < 3 && solucao_valida; i++)
                for (j = 0; j < 3 && solucao_valida; j++)
                    if (usados [resposta [block_row*3+i][block_col*3+j]] != 0)
                        solucao_valida = 0; /* Este número já apareceu. */
                    else
                        usados [resposta [block_row*3+i][block_col*3+j]] = 1;
        }

    if (!solucao_valida) {
        printf ("Resposta invalida!\n");
        return (1);
    }
    printf ("Resposta valida!\n");
    return (0);
}
```

6.

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

#define N_LINHAS 12
#define N_COLUNAS 12

#define BUSCADA "CANETAAZUL"

int main () {
    char matriz [N_LINHAS][N_COLUNAS] =
        {"CANETAAZULEO",
         "RHHEEIOYHMBE",
         "RNRYSENTOKWS",
         "IAIMITCGWVSR",
         "NHRNAURHEABK",
         "RTARUHETTNAO",
         "LMDCPNHMSATE",
         "TSEGNNSDLSHA",
         "OTWYCNLIHHRF",
         "IDRENAOBELRC",
         "LTYETOTIFCEA",
         "OFKAARATWANP"};

    int i, j, encontradas, meta;

    // Este é o tamanho da sequência buscada.
    meta = strlen (BUSCADA);

    // Para cada posição da matriz...
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_COLUNAS; j++)
        {
            // ... vê se tem a sequência começando daqui.
            if (j + meta <= N_COLUNAS) // Nem adianta procurar se extrapolar a linha.
            {
                for (encontradas = 0; encontradas < meta; encontradas++)
                    if (matriz [i][j+encontradas] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }
        }
    }

    printf ("Nao encontrou a sequencia %s\n", BUSCADA);

    return (0);
}
```

7.

```c
/* A primeira ideia é fazer 4 blocos, um para cada direção. Fica... comprido. */

#include <stdio.h>
#include <string.h>

#define N_LINHAS 12
#define N_COLUNAS 12

#define BUSCADA "CANETAAZUL"

int main () {
    char matriz [N_LINHAS][N_COLUNAS] =
        {"RHHEEIOYHMBE",
         "RCRYSENTOKWS",
         "IAILITCGWVSR",
         "NNRUAURHEABK",
         "REAZUHETTNAO",
         "ATXAADSFWERS",
         "LADAPNHMSATE",
         "TQETNNSDLSHA",
         "OQWECNLIHHRF",
         "IFRNNAOBELRC",
         "LEYATOTIFCEA",
         "XFACETAAZULA"};

    int i, j, encontradas, meta;

    // Este é o tamanho da sequência buscada.
    meta = strlen (BUSCADA);

    // Para cada posição da matriz...
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_COLUNAS; j++)
        {
            // ... vê se tem a sequência começando daqui.
            // Esquerda - direita.
            if (j + meta <= N_COLUNAS) // Nem adianta procurar se extrapolar a linha.
            {
                for (encontradas = 0; encontradas < meta; encontradas++)
                    if (matriz [i][j+encontradas] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }

            // Direita - esquerda.
            if (j - meta >= -1) // Nem adianta procurar se extrapolar a linha.
            {
                for (encontradas = 0; encontradas < meta; encontradas++)
                    if (matriz [i][j-encontradas] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }

            // Cima - baixo.
            if (i + meta <= N_LINHAS) // Nem adianta procurar se extrapolar a linha.
            {
                for (encontradas = 0; encontradas < meta; encontradas++)
                    if (matriz [i+encontradas][j] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }

            // Baixo - cima.
            if (i - meta >= -1) // Nem adianta procurar se extrapolar a linha.
            {
                for (encontradas = 0; encontradas < meta; encontradas++)
                    if (matriz [i-encontradas][j] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }
        }
    }

    printf ("Nao encontrou a sequencia %s\n", BUSCADA);

    return (0);
}
```

```c
/* Usamos agora um "truque", com offsets. Assim dá para fazer com um único bloco! */

#include <stdio.h>
#include <string.h>

#define N_LINHAS 12
#define N_COLUNAS 12

#define BUSCADA "CANETAAZUL"

int main ()
{
    char matriz [N_LINHAS][N_COLUNAS] =
        {"CANETAAZULEO",
         "RHHEEIOYHMBE",
         "RNRYSENTOKWS",
         "IAIMITCGWVSR",
         "NHRNAURHEABK",
         "RTARUHETTNAO",
         "LMDCPNHMSATE",
         "TSEGNNSDLSHA",
         "OTWYCNLIHHRF",
         "IDRENAOBELRC",
         "LTYETOTIFCEA",
         "OFKAARATWANP"};

    int i, j, i2, j2, k, encontradas, meta;

    // Este é o tamanho da sequência buscada.
    meta = strlen (BUSCADA);

    // Estes vetores são a "manha". Eles indicam as 4 direções.
    int offset_i [4] = {1,-1,0,0};
    int offset_j [4] = {0,0,1,-1};

    // Para cada posição da matriz...
    for (i = 0; i < N_LINHAS; i++)
    {
        for (j = 0; j < N_COLUNAS; j++)
        {
            // ... vê se tem a sequência começando daqui.
            for (k = 0; k < 4; k++)
            {
                if ((offset_i [k] > 0 && i+meta > N_LINHAS) || // Não pode ir para a direita.
                    (offset_i [k] < 0 && i-meta < -1) || // Não pode ir para a esquerda.
                    (offset_j [k] > 0 && j+meta > N_COLUNAS) || // Não pode ir para baixo.
                    (offset_j [k] < 0 && j-meta < -1)) // Não pode ir para cima.
                        continue;

                i2 = i;
                j2 = j;
                for (encontradas=0; encontradas < meta;
                     encontradas++, i2+=offset_i[k], j2+=offset_j[k])
                    if (matriz [i2][j2] != BUSCADA [encontradas])
                        break;

                if (encontradas == meta)
                {
                    printf ("%d %d\n", i, j);
                    return (0); // Pode parar!
                }
            }
        }
    }

    printf ("Nao encontrou a sequencia %s\n", BUSCADA);

    return (0);
}
```