9.1 Implemente uma função `void ordenar(char str[])` que ordena os carateres numa cadeia pelos seus códigos. Por exemplo: se `str = "ALGORITMO"` então após execução devemos ter `str = "AGILMOORT"`.

Sugestão: cadeias de carateres em linguagem C são variáveis indexadas, pelo que podemos usar um dos algoritmos de ordenação apresentados nas aulas teóricas (seleção, inserção ou "quicksort"). Pode determinar o número de carateres da cadeia usando `strlen`.

▷ **9.2** Escreva uma função `int anagramas(char str1[], char str2[])` que determina se duas cadeias de carateres são *anagramas*, isto é, se se escrevem com os mesmos carateres. O resultado deve ser 1 em caso afirmativo e 0 caso contrário. Por exemplo, "deposit" e "topside" são anagramas:

```
char str1[] = "deposit";
char str2[] = "topside";
int r = anagramas(str1,str2); // resultado 1
```

Sugestão: use uma função auxiliar como no exercício anterior para ordenar ambas as cadeias; as cadeias originais são anagramas se e só se as cadeias ordenadas são *exatamente iguais* carater-a-carater.

9.3 O conceito de anagrama pode ser aplicado mais geralmente a uma *frase* com várias palavras, ignorando os espaços, sinais de pontuação e a distinção entre maiúscula e minúsculas. Por exemplo a pergunta em Latim "`Quid est veritas?`" (*O que é a verdade?*) é um anagrama de "`Est vir qui adest`" (*É o homem que aqui está*). Modifique a programa do exercício anterior para testar anagramas neste sentido mais lato.

Sugestão: começe por escrever uma função auxilar `void normalizar(char str[])` para "normalizar" uma cadeia de carateres eliminando todos os carateres não-letra e convertendo todas as letras em minúsculas.

9.4 O programa para *Quicksort* apresentado na aula teórica escolhe sempre o primeiro valor da sub-sequência para *pivot*. Isto causa que a complexidade no caso em a sequência está ordenada seja quadrática.

Pretende-se melhorar esta escolha usando como *pivot* a *mediana* do primeiro, último e do ponto-médio da sub-sequência (ver exercício 3.8). Para tal basta trocar a mediana com o valor no ínicio da partição:

$\mathsf{partition}(l, u)$:
    $j \leftarrow$ índice da mediana de $v[l], v[u], v[(l+u)/2]$
    trocar $v[l], v[j]$
    continua como anteriormente...

Complete e implemente esta modificação no programa do *Quicksort*.

9.5 Defina uma função `int identidade(int mat[N][N])` para verificar se uma matrix $N \times N$ é a *identidade* (isto é, contém 1 na diagional principal e 0 nas outras posições).

O resultado deve ser 1 ou 0 conforme a matriz é ou não identidade. A sua função deve funcionar para qualquer dimensão N declarada como constante usando usando `#define`.

9.6 Defina duas funções

```
int soma_diagonal1(int mat[N][N]);
int soma_diagonal2(int mat[N][N]);
```

que dadas matrizes $N \times N$ soma: (1) os valores na diagonal principal (isto, tais que índices i, j são iguais); (2) os valores na diagonal secundária (isto é, tais que os índices satisfazem $i + j = N - 1$). As funções devem funcionar para qualquer dimensão N declarada como constante usando `#define`.

▷ **9.7** Um *quadrado mágico* é uma matriz quadrada de números inteiros tal que todas as linhas, colunas e diagonais somam o mesmo valor. No exemplo seguinte cada linha, coluna e diagonal soma o mesmo valor (15 neste caso):

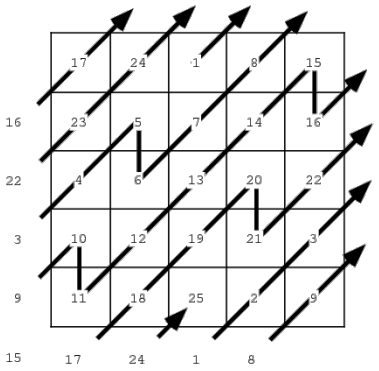
$$\begin{bmatrix} 2 & 7 & 6 \\ 9 & 5 & 1 \\ 4 & 3 & 8 \end{bmatrix}$$

Escreva uma função `int magico(int a[20][20], int n)` que testa se uma matriz é um quadrado mágico. A matriz é representada por uma variável indexada `a` com dimensão declarada 20×20; o argumento `n` indica qual sub-matriz a considerar: por exemplo, se `n = 3` devemos testar se a sub-matriz de 3×3 é quadrado mágico (e ignorar o resto da matriz). O resultado deve ser um inteiro: 1 se é quadrado mágico e 0 caso contrário.

9.8

O algoritmo seguinte permite gerar quadrados mágicos de tamanho $n \times n$ para n ímpar: começemos por colocar um "1" no meio da primeira linha; em seguida, vamos colocar os números de 2 a n^2: o próximo numero é colocado na posição na diagonal para cima e para a direita se esta se encontrar vazia; se a posição já se encontra preenchida, então passamos para a posição em baixo. Em qualquer dos casos, o movimento é feito considerando que os bordos cima/baixo e esquerdo/direito do quadrado estão ligados (isto é, se sairmos pela direita re-entramos na posição correspondente à esquerda e se sairmos por cima re-entramos na posição corresponde por baixo).

A figura seguinte ilustra a geração de um quadrado mágico 5×5.



Escreva um programa que constroi um quadrado mágico de dimensão impar < 50 usando este algoritmo.