8.1 Defina uma função `void eliminar(char str[], char ch)` que elimina a primeira ocorrência de um carater `ch` de uma cadeia de carateres. Exemplo: se `str = "ABBA"`, então depois de executar `eliminar(str, 'B')` devemos ter `str = "ABA"`.

Tenha o cuidado de colocar corretamente o terminador `'\0'`.

▷ **8.2** Escreva uma função `int ordenada(int vec[], int size)` que testa se uma variável indexada de inteiros está por *ordem ascendente* em sentido lato, isto é, se $vec[i] \leq vec[i+1]$ para todos os índices i de 0 a $size - 2$. O resultado deve ser 1 em caso afirmativo e 0 em caso negativo. A função não deve modificar os valores da variável indexada.

Exemplos: se `vec = {1, 3, 3, 5, 6}` então `ordenada(vec, 5)` deve retornar 1; se `vec = {1, 3, 2, 5, 6}` então `ordenada(vec, 5)` deve retornar 0.

8.3 Escreva uma função `int desordem(int vec[], int size)` que conta quantos pares de valores numa variável indexada estão fora de ordem, isto é, $vec[i] > vec[i+1]$. Exemplo: se `vec = {3, 1, 2, 2, 4, 0}` e `size=6` então o resultado deve ser 2 (porque $3 \not\leq 1$ e $4 \not\leq 0$).

Note ainda que se a sequência estiver por ordem ascendente, então o resultado é 0 e se estiver por ordem descente, então o resultado é `size` $- 1$.

8.4 Re-utilizando os algoritmos apresentados na aula 17 (**ordenação por seleção** e **por inserção**), escreva um programa completo que lê uma sequência de inteiros positivos da entrada-padrão terminada por zero, ordena e imprime por ordem crescente.

Sugestão: a função `main` no seu programa deve declarar uma variável indexada com um tamanho máximo (por exemplo: 1000), ler os valores, invocar a função função de ordenação e imprimir a sequência final ordenada. Experimente ambos os algoritmos (deve obter resultados iguais).

8.5 Modifique a função que implementa **ordenação por seleção** apresentada na aula 16 para ordenar por ordem decrescente.

8.6 Defina uma função `int segundo_menor(int vec[], int size)` que encontra o segundo menor valor de uma variável indexada `vec` com `size` elementos. Pode assumir que `size`≥ 2.

Sugestão: para encontrar o segundo menor valor basta efetuar as duas primeiras iterações do algoritmo de **ordenação por seleção** apresentado na aula 16. A sua função pode modificar a ordem dos elementos de `vec`.

▷ **8.7** Defina uma função `void sort_desc(int vec[], int n)` que ordena um vetor de inteiros de comprimento n por *ordem descendente* (isto é, primeiro o maior valor, depois o segundo maior, e assim sucessivamente). Por exemplo: se `vec={3,2,1,3,5}` e `n=5` então depois de executar `sort_desc(vec, n)` devemos ter `vec={5,3,3,2,1}`.

Sugestão: adapte um dos algoritmos de ordenação (*seleção* ou *inserção*) apresentados nas aulas 18 e 19 para ordenar por ordem inversa.

8.8 Considere a primeira versão do algoritmo de Euclides para calcular o máximo divisor apresentado na aula 7:

```
int mdc(int a, int b)  {
  while (a != b) {
    if(a > b)
      a = a - b;
    else
      b = b - a;
  }
  return a;
}
```

Acrescente uma asserção que exprime as pré-condições para que este algoritmo termine: a, b devem ser ambos positivos.

8.9 Considere a função que calcula a mediana de três valores a, b, c (Exercício 3.8). Modifique a sua solução acrescentando uma asserção correspondente à pós-condição seguinte: seja r o resultado de mediana(a, b, c); então $\min(a, b, c) \leq r \leq \max(a, b, c)$.

Sugestão: defina funções auxiliares para calcular o mínimo e o máximo dos três valores.

8.10 Considere a função pedida no exercício 7.6 (converter uma cadeia de algarismos decimais para um inteiro). Acrescente asserções para exprimir a pré-condição que todos os carateres da cadeia são algarismos decimais (i.e. carateres entre `'0'` e `'9'`).

8.11 Considere a seguinte função para inserir um valor num vector de n elementos baseada no algoritmo usado na *ordenação por inserção*; assumimos que os valores do vetor estão por ordem ascendente à entrada da função; no final, o vetor terá mais um valor mantém a ordem ascendente.

```
void inserir(int vec[], int n, int x) {
    int j = n-1;
    while(j>=0 && vec[j]>x) {
      vec[j+1] = vec[j];
      j--;
    }
    vec[j+1] = x;
}
```

Acrescente asserções à função para exprimir as seguintes condições de correção:

- uma *pré-condição* à entrada da função: o vetor está ordenado, ou seja, vec$[i] \leq$ vec$[i+1]$ para todos os índices $0 \leq i < n$;
- uma *pós-condição* à saida da função: o vetor continua ordenado e o comprimento aumentou 1 unidade, ou seja, vec$[i] \leq$ vec$[i+1]$ para todos os índices $0 \leq i \leq n$