5.1 Usando a função de teste de primalidade apresentada na aula 7, escreva um programa que imprime uma lista de primos até um limite superior especificado pelo utilizador. Exemplo:

```
Limite superior? 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

5.2 Modifique a implementação do algoritmo de Euclides usando subtrações sucessivas (aula teórica 7) para imprimir uma linha de texto com os valores dos inteiros a, b em cada iteração; no final deve ainda imprimir o m.d.c. e o número de iterações efetuadas. Exemplos para `mdc(12,18)` e `mdc(36,21)`:

```
mdc(12,18) = mdc(12,6) = mdc(6,6) = 6
3 iterações
mdc(36,21) = mdc(15,21) = mdc(15,6) = mdc(9,6) = mdc(3,6) = mdc(3,3) = 3
6 iterações
```

5.3 Considere a implementação em C do algoritmo de Euclides usando subtrações sucessivas apresentada na aula teórica 7. Simule a execução passo-a-passo de `mdc(52,0)`. Porque é que o programa não termina neste caso?

Sugestão: observe cuidadosamente a justificação matemática para a terminação do algoritmo e identifique qual a hipótese que não se verifica.

5.4 Indique qual o menor dos tipos númericos `short`, `int` ou `long` é suficiente para a armazenar as seguintes quantidades; assuma os limites na arquitetura X86 demonstrados na aula teórica 8.

(a) número de dias num ano;

(b) número de horas num ano;

(c) número de segundos num dia;

(d) número de segundos num mês (31 dias);

(e) número de segundos desde 1 de janeiro de 1900.

5.5 Escreva um programa que lê repetidamente carateres até encontrar uma mudança de linha (`\n`) e contabiliza o *número total de letras* (isto é, carateres de `'A'` a `'Z'` e de `'a'` a `'z'`). Exemplo (em sublinhado o texto introduzido pelo utilizador):

```
Ola, Mundo!
A frase contém 8 letra(s)
```

Sugestão: usar `getchar()` para ler um carater de cada vez.

▷ **5.6** Escreva um programa para contar palavras da entrada-padrão. Considere que as palavras são sequências de "carateres normais" (e.g. letras, algarismos ou sinais de pontuação) separados por "carateres brancos" (espaços, tabulação ou mudanças de linha, ou seja, `' '`, `'\t'` ou `'\n'`).
Por exemplo, o texto seguinte contém 13 palavras (note que a sequência `---` é considerada uma palavra):

```
To be or   not to be,  that is the question.
--- William Shakespeare
```

Sugestão: Utilize duas variáveis para guardar os dois últimos carateres lidos; podemos então contar o número de vezes que um carater "normal" (i.e., não branco) é seguido de um carater branco. No exemplo seguinte assinalamos as transições relevantes com o simbolo `^`:

```
To␣be␣or␣␣␣␣not␣to␣be␣
  ^  ^  ^     ^  ^  ^
```

5.7 No jogo *SCRABBLE* os jogadores pontuam formando palavras do dicionário com fichas representando letras individuais. A pontuação de cada letra depende da sua raridade no dicionário. As pontuações para o inglês são: A,E,I,L,N,O,R,T,S,U: 1 ponto; D,G: 2 pontos; B,C,M,P: 3 pontos; F,H,V,W,Y: 4 pontos; K: 5 pontos; J,X: 8 pontos; Q,Z: 10 pontos. A pontuação duma palavra é a soma dos pontos de letras individuais.[1] Por exemplo: a palavra "PITFALL" vale $3+1+1+4+1+1+1=12$ pontos.

Escreva um programa que lê uma sequência de carateres de uma palavra terminada por `EOF` e calcula e imprime a sua pontuação; considere que espaços ou outros carateres não-letras valem 0 pontos.

Sugestão: defina uma função auxiliar para calcular a pontuação para um carater apenas; pode ainda usar a instrução `switch` para selecionar o a pontuação de cada letra.

▷ **5.8** Pretende-se calcular o *logaritmo natural* (isto é, de base e) usando a série de Taylor:

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}$$

Escreva uma função `double serie_log(double x, int n)` que calcula aproximamente a série acima somando os termos até à potência n de x. Por exemplo: `serie_log(x, 3)` deve calcular $x - x^2/2 + x^3/3$. Pode assumir que $n \geq 1$. Tenha o cuidado de evitar o cálculo desnessário de potências sucessivas de x.

5.9 A fórmula de Gregory-Leibniz para aproximar π é:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \cdots\right) = 4 \times \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1}$$

Implemente a função `double aprox_pi(int n)` que calcula π aproximadamente somando os primeiros `n` termos desta série.

Esta série converge lentamente; pode constatar isso escrevendo um programa principal que compare as aproximações obtidas com 10, 100 e 1000 termos com a constante `M_PI` definida no *header* `math.h`.

[1] Há outras condições que afetam a pontuação que vamos ignorar neste exercício.