

3.1 Considere o programa para calcular comissões de transferência (ver slides). Acrescente um ciclo ao programa de forma a que este leia repetidamente valores e imprima as comissões correspondentes; deve terminar quando for introduzido o valor 0.

3.2 No nosso calendário Gregoriano os *anos bissextos* têm 366 dias em vez dos normais 365. As regras para determinar se um ano é bissexto são:

- São bissextos todos os anos múltiplos de 400 (e.g., 1600, 2000, 2400, 2800...);
- São bissextos todos os múltiplos de 4, exceto se forem múltiplos de 100 mas não de 400 (e.g., 1996, 2000, 2004, 2008, 2012, 2016...);
- Não são bissextos todos os demais anos.

Defina uma função `int bissexto(int n)` que verifica estas condições para um ano `n`; o resultado deve ser 1 se o ano for bissexto e 0 caso contrário.

▷ **3.3** Escreva uma função `int prox_bissexto(int n)` que calcula o próximo ano bissexto a partir do ano dado; se `n` for um ano bissexto, então o resultado deve ser `n`. Exemplos: `prox_bissexto(1980)` retorna 1980; `prox_bissexto(2017)` retorna 2020.

Sugestão: utilize um ciclo `while` começando no ano dado e a função do exercício 3.2 para determinar se um ano é bissexto.

3.4 Generalizando o exemplo dos slides, escreva um programa que lê um inteiro não-negativo e imprime os seus dígitos binários. Podemos obter a representação de um inteiro positivo n numa base b fazendo divisões sucessivas por b até obter quociente 0; os restos das divisões são os “dígitos” (de 0 a $b - 1$).

Exemplo: obtemos a representação em binário de 25 fazendo 4 divisões sucessivas por 2:

$$\begin{array}{ccccccccc}
 25 & \xrightarrow{/2} & 12 & \xrightarrow{/2} & 6 & \xrightarrow{/2} & 3 & \xrightarrow{/2} & 1 & \xrightarrow{/2} & 0 \\
 \downarrow \%2 & & \downarrow \%2 & & \downarrow \%2 & & \downarrow \%2 & & \downarrow \%2 & & \\
 1 & & 0 & & 0 & & 1 & & 1 & &
 \end{array}$$

Assim, 25 decimal é representado em binário por 11001. Note que os dígitos são obtidos pela ordem do menos significativo para o mais significativo.

▷ **3.5** Pretende-se calcular o *logaritmo natural* (isto é, de base e) usando a série de Taylor:

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

Escreva uma função `double serie_log(double x, int n)` que calcula aproximadamente 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 desnecessário de potências sucessivas de x .

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

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

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`.

3.7 Usando a função de teste de primalidade dos slides, 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

3.8 Modifique a implementação do algoritmo de Euclides usando subtrações sucessivas (ver slides) 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`

3.9 Considere a implementação em C do algoritmo de Euclides usando subtrações sucessivas (ver slides). 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.

3.10 Indique qual o menor dos tipos numéricos `short`, `int` ou `long` é suficiente para a armazenar as seguintes quantidades; assuma os limites na arquitetura X86.

- (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.