1 - Implemente a seguinte função para retornar o equivalente em Fahrenheit de uma temperatura Celsius.

$$F = \frac{9}{5}C + 32$$

Use esta função para escrever um código que imprima uma tabela mostrando os equivalentes em Fahrenheit de todas as temperaturas Celsius de 0 a 100 graus. Use uma posição de precisão à direita do ponto decimal para os resultados. Imprima as saídas em um formato tabular limpo que minimize o número de linhas de saída enquanto permanece legível.

2 - Um número inteiro maior que 1 é considerado primo se for divisível por apenas 1 e ele próprio. Por exemplo, 2, 3, 5 e 7 são números primos, mas 4, 6, 8 e 9 não são.
a) Escreva uma função que determine se um número é primo.
b) Use esta função em um código que determina e imprime todos os números primos entre 2 e 1.000.
c) Inicialmente, você pode pensar que n / 2 é o limite superior para o qual você deve testar se um número é primo, mas você precisa ir apenas até a raiz quadrada de n. Reescreva o código e execute-o nas duas maneiras para mostrar que você obtém o mesmo resultado.

3 - Um número inteiro é considerado um número perfeito se a soma de seus fatores, incluindo 1 (mas não o número em si), for igual ao número. Por exemplo, 6 é um número perfeito, porque 6 = 1 + 2 + 3. Escreva a função perfect que determina se o número do parâmetro é um número perfeito. Use esta função em um código que determina e imprime todos os números perfeitos entre 1 e 1000. Imprima os fatores de cada número perfeito para confirmar que o número é realmente perfeito. Desafie o poder do seu computador testando números muito maiores que 1000.

4 - Os computadores estão desempenhando um papel crescente na educação. Escreva um programa que ajude um aluno do ensino fundamental a aprender multiplicação. Use o módulo aleatório para produzir dois números inteiros positivos de um dígito. O programa deve exibir uma pergunta, como
Quanto é 6 vezes 7?
O aluno digita a resposta. Em seguida, o programa verifica a resposta do aluno. Se estiver correto, imprima a string "Muito bom!" na tela e faça outra pergunta de multiplicação. Se a resposta for errada, exibir "Não. Por favor, tente novamente." e deixe o aluno tentar a mesma pergunta repetidamente até que o aluno finalmente acerte. Uma função separada deve ser usada para gerar cada nova pergunta. Este método deve ser chamado uma vez quando o programa iniciar a execução e sempre que o usuário responder à pergunta corretamente.

5 - Escreva um código que reproduza o jogo de “adivinhar o número” da seguinte forma:
Seu programa escolhe o número a ser adivinhado selecionando um número inteiro aleatoriamente no intervalo de 1 a 1000. O programa exibe
Eu tenho um número entre 1 e 1000.

Você consegue adivinhar o meu número?
Digite seu primeiro palpite.
O jogador digita um primeiro palpite. O programa responde com um dos seguintes:

1. Excelente! Você adivinhou o número! Você gostaria de jogar novamente (s ou n)?
2. Muito baixo. Tente novamente.
3. Muito alto. Tente novamente.

Se o palpite do jogador estiver incorreto, seu programa deve repetir até que o jogador finalmente obtenha o número correto. Seu programa deve continuar dizendo ao reprodutor Muito alto ou Muito baixo para ajudar o leitor a se concentrar na resposta correta. Após o término do jogo, o programa deve solicitar ao usuário que digite "s" para jogar novamente ou "n" para sair do jogo.

6 - (Torres de Hanói) Todo programador iniciante deve lidar com certos problemas clássicos. As Torres de Hanói (veja a figura abaixo) são uma das mais famosas. Diz a lenda que, em um templo no Extremo Oriente, os monges estão tentando mover uma pilha de discos de uma estaca para outra. A pilha inicial tinha 64 discos enfiados em uma estaca e organizados de baixo para cima, diminuindo o tamanho. Os monges estão tentando mover a pilha dessa estaca para uma segunda estaca, sob as restrições de que exatamente um disco é movido por vez e que em nenhum momento um disco maior pode ser colocado acima de um disco menor. Uma terceira estaca está disponível para a retenção temporária de discos. Supostamente, o mundo terminará quando os sacerdotes concluírem sua tarefa; portanto, há pouco incentivo para facilitarmos seus esforços. Vamos supor que os padres estão tentando mover os discos da estaca 1 para a estaca 3. Desejamos desenvolver um algoritmo que imprima a sequência precisa das transferências de disco estaca-estaca.

Se abordássemos esse problema com métodos convencionais, rapidamente nos encontraríamos irremediavelmente envolvidos no gerenciamento dos discos. Em vez disso, se atacarmos o problema com a recursão em mente, ele se tornará imediatamente tratável.

Mover n discos pode ser visualizado em termos de mover apenas $n - 1$
discos (daí a recursão), da seguinte maneira:
a) Mova $n - 1$ discos da estaca 1 para a estaca 2, usando a estaca 3 como uma área de retenção temporária.
b) Mova o último disco (o maior) da estaca 1 para a estaca 3.
c) Mova os discos $n - 1$ da estaca 2 para a estaca 3, usando a estaca 1 como uma área de retenção temporária.
O processo termina quando a última tarefa envolve mover disco n = 1, ou seja, o caso base. Isso é feito trivialmente movendo o disco sem a necessidade de uma área de retenção temporária.

Escreva um programa para resolver o problema das Torres de Hanoi. Use uma função recursiva com quatro parâmetros:
a) O número de discos a serem movidos
b) A estaca na qual esses discos são inicialmente enfiados
c) A estaca para o qual essa pilha de discos deve ser movida

d) A estaca a ser usada como área de retenção temporária
Seu código deve imprimir as instruções precisas necessárias para mover os discos do ponto inicial para o ponto de destino. Por exemplo, para mover uma pilha de três discos da estaca 1 para a estaca 3, seu programa deve imprimir a seguinte série de movimentos:
1 → 3 (Isto indica mover um disco da estaca 1 para a estaca 3)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3