

1. Un algoritmo para saber si un número es divisible por 7 consiste en multiplicar por 2 el dígito de las unidades y el resultado se resta al número que forman las cifras restantes. Si el resultado es divisible por siete, el número original lo es. Este proceso se repite hasta llegar a un número de un solo dígito (puede ser negativo). El número original es divisible por siete si se obtiene -7, 0, o 7. Realice una función que usando este algoritmo retorne verdadero o falso indicando la divisibilidad del argumento recibido. El prototipo de la función deberá ser:

```
bool esDivisiblePor7(int nro);
```

2. El algoritmo de Euclides permite calcular el máximo común divisor entre 2 números en forma muy eficiente. Se basa en que si tenemos 2 números enteros A y B, con  $A \geq B$ , el MCD entre A y B es equivalente al MCD entre B y el resto de la división entera entre A y B ( $\text{MCD}(A, B) == \text{MCD}(B, A \% B)$ ). (Hint: el MCD entre cualquier número X y cero es X).

Una relación muy sencilla entre el MCD y el MCM es:

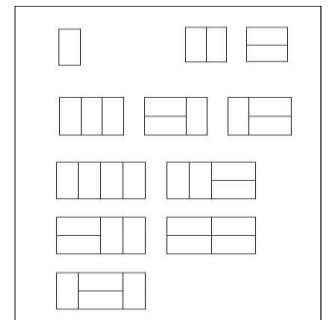
$$\text{MCM}(a, b) = (a \cdot b) / \text{MCD}(a, b)$$

Realizar funciones que realicen el cálculo del MCD y del MCM de dos números.

3. Se desea averiguar de cuantas formas posibles se puede llenar un rectángulo de  $2 \times n$  casilleros utilizando fichas de dominó (de  $2 \times 1$ ). En la figura se ve que para  $n = 1$  hay una sola forma, para  $n = 2$  hay 2 formas, para  $n = 3$  hay 3 formas y para  $n = 4$  hay 5 formas.

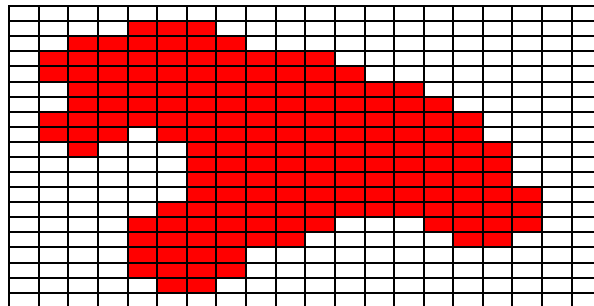
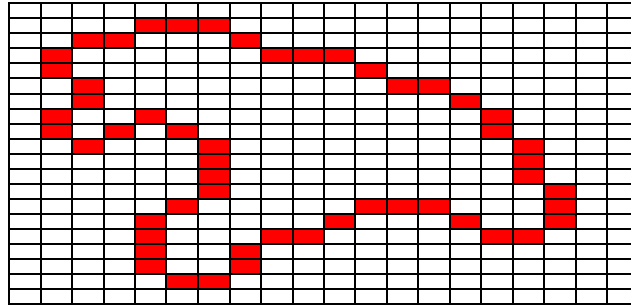
Escribir un programa que calcule la cantidad de formas de llenar con dominós un rectángulo de  $2 \times n$  casilleros.

**Hint:** Descomponga el problema en combinaciones que empiecen con una ficha vertical y combinaciones que empiecen con fichas horizontales.



4. Implementar una función que resuelva el problema de las torres de Hanoi de N discos. Encontrar la cantidad de movimientos que hay que resolver el problema en función de N. Suponga que cuenta con una máquina capaz de mover 1 millón de discos por segundo. ¿Qué tiempo le llevaría resolver el problema original de 99 discos?

5. Dada una matriz  $M[MX][MY]$  que representa el valor de cada píxel de un rectángulo de  $MX \times MY$ , y asumiendo que todo el rectángulo tiene el mismo color C, excepto una curva cerrada de color C1, se desea hacer una función que, partiendo de un punto interior a la curva cerrada, pueda pintar todo el interior con el color C1



Se solicita implementar la función:

```
void FloodFill(int M[MX][MY], int X, int Y, int C1);
```

Que recibe la matriz de píxeles M, (MX x MY elementos), las coordenadas de un punto (X,Y) interior a la curva cerrada, y el color del contorno C1 (con el cual deberá llenar su interior).

6. Implemente la función **Replace** que reemplace en el string nativo **s** todas las apariciones del carácter **viejo** por el carácter **nuevo**.

Prototipo:

```
void Replace(char s[], char nuevo, char viejo);
```

7. Implementar la función **StrStr** que reciba dos strings nativos y busque sobre el primero de ellos por la ocurrencia del segundo.

Prototipo:

```
int StrStr(char s1[], char s2[]);
```

Como retorno, **StrStr** deberá retornar el índice del elemento sobre **s1** donde comienza la ocurrencia de **s2**. Si **s2** no aparece en **s1**, la función deberá retornar **-1**.

Ejemplos:

```
StrStr("JUAN ESTA CASADO CON MARIA", "ASADO") retornará 11.  
StrStr("ABCDE", "BCE") retornará -1.
```

8. Implementar la función `StrStr` del ejercicio anterior pero utilizando `std::string`.

Prototipo:

```
int StrStr(const string &s1, const string &s2);
```

Hint: `s1.find(s2)`

9. Hacer un programa que lea un archivo con 2 columnas de valores en punto flotante representando puntos (x,y) y diga para que x el valor de y es máximo. (peaks.dat).

10. Se desea realizar un programa que pueda leer palabras desde el dispositivo estándar de entrada (cin) y arme estadísticas acerca de las palabras distintas y cantidad de veces que se repite cada una. Como resultado, el programa deberá imprimir cada palabra con las veces que ésta se repite. Para ello se diseñan las siguientes estructuras:

```
struct StatWords {  
  
    struct StatRecord{  
        string word;           // palabra en si  
        unsigned int repCnt;    // cantidad de repeticiones  
    };  
    vector<StatRecord> records;  
  
    void addWord(const string &word); // agrega una palabra.  
                                        // si ya está presente en records  
                                        //      -> incrementa su repCnt  
                                        // si no  
                                        //      -> agrega un registro nuevo en records  
  
    void print();                  // imprime las palabras y cuantas veces se  
                                    // repite  
};
```

Se solicita implementar los métodos `addWord` y `print`. Un posible programa para chequear el programa podría ser:

```
int main()  
{  
    StatWords stats;  
    string w;  
  
    while (cin >> w)  
        stats.addWord(w);  
  
    stats.print();  
    return 0;  
}
```

Probar el programa ejecutándolo y redirigiendo el dispositivo standard de entrada desde un archivo.