Objetivo de la práctica:
Utilizar el paso de parámetros por referencia y las funciones recursivas.
Nota: Todos los ejercicios son obligatorios a excepción del último (T. Hanoi).

Funciones con parámetros por referencia

- Las funciones, por norma general, resuelven algún subproblema requerido desde el programa principal o desde otra función. Para ello, en la mayoría de ocasiones, necesitarán **recibir parámetros** (a modo de variables) desde el programa que las invoca, de forma que puedan operar con ellos y conseguir su objetivo. **El paso de estos parámetros/variables** puede implementarse en C++ de dos formas distintas:

    - En primer lugar, todo sub/programa que llama a una función puede enviarle los parámetros necesarios por **valor**. En este caso, la función al recibir los parámetros se hará una **copia** de los mismos y operará con esta copia, de modo que, las variables del sub/programa principal utilizadas como parámetros no se verán afectadas. Por esta razón se dice que los parámetros por valor son *parámetros de entrada.*

    - En segundo lugar, en C++ se pueden pasar parámetros/variables por **referencia**. En este caso, se utilizan las mismas variables (direcciones de memoria) tanto en el programa que llama a la función como en la propia función misma (es decir, no se hacen copias), por lo que si modificamos el valor de algún parámetro/variable dentro de la función, éste se mantendrá cambiado al terminar la función. (*parámetros de salida*). Esta modalidad de paso de parámetros suele utilizarse para implementar funciones que devuelvan más de un valor (el valor de retorno de la función + las variables por ref. definidas).

    Para diferenciar ambos tipos de parámetros (ya que ambos pueden aparecer indistintamente en la cabecera de la función), **los parámetros por referencia** utilizan el símbolo **&**, tanto en el prototipo como en la cabecera de la función.

```
Ej: <tipo> funcion (<tipo_p> &param); // Parámetro param por referencia
Ej: <tipo> funcion (<tipo_p> param);  // Parámetro param por valor
```

SOBRE LA PRÁCTICA ANTERIOR:

Prueba el programa en el fichero **`ejemplo_pract5_vocales_con_errores.cpp`** (accesible en la página web de la asignatura); si te parece que no funciona correctamente modifícalo hasta que funcione como debería.

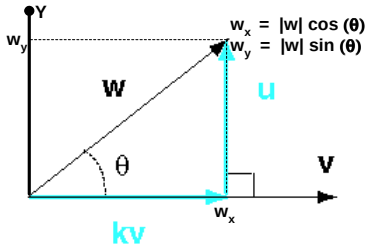
Problemas:

1. Escribe otra función que dado un vector 2D y un ángulo α, devuelva el vector girado α grados en sentido antihorario .

2. Escribe una función que calcule el vector **kv**, es decir, la proyección del vector **w** sobre el **v** (ver figura), ambos recibidos como parámetros de entrada. Los vectores 2D pueden representarse a partir de sus **componentes x e y**, por ejemplo, el vector **w** será tratado en nuestro programa como **(w_x, w_y)**

Pasos:

a. Calcular las longitudes de los vectores:
   $|\mathbf{w}| = \mathbf{sqrt}\ (w_x * w_x + w_y * w_y);$
   $|\mathbf{v}| = \mathbf{sqrt}\ (\ldots)$
b. Obtener los vectores unitarios (por componentes):
   $w_u = w / |w|$
   $v_u = v / |v|$
c. Calcular el producto escalar (coseno del ángulo abierto en
   $w_u \cdot v_u = \cos\theta = (w_{ux} * v_{ux} + w_{uy} * v_{uy})$
d. Resolver la ecuación (por componentes)
   $kv = |w| (w_u \cdot v_u) v_u$



3. Escribe una función que genere combinaciones de 4 números sin repetición. El dominio de los números puede fijarse del 1 … 9. Sugerencia: Definir la función `int aleatorio(int ini, int fin)` para generar números aleatorios entre `ini` y `fin`.

FUNCIONES RECURSIVAS

Dentro del código de una función recursiva siempre hay una sentencia o expresión donde aparecerá la **llamada a la misma función**. Para no generar infinitas llamadas (dado que la función se llama a sí misma) necesitamos tener en cuenta lo siguiente:

- Una función recursiva resolverá un problema de talla N, considerando resuelto el problema de talla N – 1. Por tanto, **los parámetros de la función deben variar** de una llamada a otra.

ej: `factorial(N) = N * factorial (N-1);  // problema_de_talla_N = N * problema_de_talla_N-1`

- **Antes** de que se produzca la recursión, debe aparecer la **condición de parada** (caso base), que estará relacionada con alguno de los parámetros de la función. Al detectar este caso, no se producirán más llamadas recursivas y se devolverá el valor correpondiente al caso sencillo o base (ej: `factorial(0) = 1`)

PROBLEMAS (funciones recursivas)

4. Escribe funciones recursivas para las siguiente operaciones entre 2 enteros:

   a. Sumar: Dentro de la función sólo se podrán utilizar incrementos unitarios, por tanto, **no se puede utilizar el operador suma**.
   b. Multiplicar: En este caso sólo se podrá utilizar el operador suma, y **no el de multiplicación**.
   c. La función exponencial, x^y siendo x, y números enteros ($x^y = x * x^{y-1}$).

5. Escribe una función recursiva para calcular el número de combinaciones (sin repetición) de n objetos tomados de k en k, siendo n > 1 y 0 < k < n:

```
combi(n,k) = 1;                                    si k = 0 ó k = n
```

```
combi(n,k) = combi(n-1, k) + combi(n-1, k-1);    en
otro caso
```
[1]

Por ejemplo: Sean n personas, queremos determinar cuántos grupos de k personas se pueden formar de entre éstas.

ej.: 4 personas (A, B, C y D) … cuantos grupos de 2 se pueden formar?

6. Escribe 2 funciones (recursión indirecta) para calcular si un número es par o impar:

```
bool par (int a);
bool impar (int a);
```

Nota: La naturaleza de este problema revela la necesidad de introducir ***recursión indirecta***, por ejemplo, la función `par` sobre un numero `a > 1` devolverá cierto si el número `(a-1)` es impar; La función `impar` actuará de un modo equivalente con el resultado de la función `par(a-1);` El uso del **operador módulo** está (lógicamente) **prohibido** para este problema.

7. Escribe un función recursiva que, dada una frase la muestre en orden inverso, ejemplo:

```
Ingeniería informática >>>> acitámrofni aíreinegnI
```

Sugerencia: puedes leer carácteres recursivamente (de uno en uno) hasta detectar el cado base (\n), en ese momento parar la recursión y al volver imprimir los valores leídos (el orden será el inverso al introducido).

LA LEYENDA DE LAS TORRES DE HANOI (Opcional)

Dice la leyenda que, al crear el mundo, Dios situó sobre la Tierra tres varillas de diamante y sesenta y cuatro discos de oro. Los discos son todos de diferente tamaño e inicialmente fueron colocados en orden decreciente de diámetros sobre la primera de las varillas. También creó Dios un monasterio cuyos monjes tienen la tarea de trasladar todos los discos desde la primera varilla a la tercera. La única operación permitida es mover un disco de una varilla a otra cualquiera, pero con la condición de que no se puede situar encima de un disco otro de diámetro mayor. La leyenda dice también que cuando los monjes terminen su tarea, el mundo se acabará. Implementa la función recursiva que calcule los movimientos necesarios para mover N discos de acuerdo con las reglas anteriores y rellena la tabla siguiente:

```
void hanoi(int N, int desde, int hacia, int usando)
```

Número de Discos:	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Número de Movimientos:	**1**	**3**					

[1] El cálculo del número de combinaciones de n elementos tomados de k en k se puede descomponer en dos problemas: a) calcular el número de combinaciones de n-1 elementos tomados de k en k y b) el número de combinaciones de n-1 elementos tomados de k-1 en k-1.

- Suponiendo que los monjes son capaces de realizar un movimiento por segundo, ¿sabrías calcular en cuánto tiempo acabará el mundo, según la leyenda?.

Solución:

Siendo $\boldsymbol{m_k}$ *el número mínimo de movimentos para pasar* ***k*** *discos de una torre a otra, tenemos que:*

$m_1 = 1$, $m_2 = 3$, $m_3 = 7$, $m_4 = 15$, $m_5 = 31$ *(datos extraídos de forma empírica de la tabla anterior …)*

Es fácil observar un par de cosas:

- $\boldsymbol{m_k = 2*m_{k-1} + 1}$.
- $\boldsymbol{m_k = 2^k - 1}$. *A partir de lo anterior y mediante el método de inducción, puede hacerse una demostración fácil de esta fórmula.*

Como son 64 discos, el número de movimientos es $\boldsymbol{2^{64} - 1 = 18446744073709551616}$. *Si suponemos que los monjes tienen la suficiente habilidad como para hacer un movimiento en un segundo, en un día harán 60*60*24 movimientos. Y en un año de 365 días: 60*60*24*365. Dividimos el número de movimientos por el resultado de la operación anterior y nos debe dar, aproximadamente, medio billón de años. Sólo falta averiguar cuantos años se estiman que el hombre lleva sobre la tierra y sabremos el tiempo que le queda sobre ella.*