Arquitectura de las Computadoras.
Plancha 1 - 2016.
C y sistemas de numeración posicional.

1) Utilizando corrimiento, operadores de bits y operaciones enteras complete para que la igualdad valga

```
10000000 00000000 00000000 00000000  == ....   << ...

10000000 00000000 10000000 00000000  == (1 << ...)  | (1 << ...)

11111111 11111111 11111111 00000000  == -1 & .....

10101010 00000000 00000000 10101010  == 0xaa ..... (0xaa << .......)

00000000 00000000 00000101 00000000  == ( 5 ..... 8)

11111111 11111111 11111110 11111111  == -1 & (.... (1 << 8))
```

2) Implementar una función que tome tres parámetros *a*, *b* y *c* y que rote los valores de las variables de manera que al finalizar la función el valor de *a* se encuentre en *b*, el valor de *b* en *c* y el de *c* en *a*. ¿Puede implementarse esta función sin utilizar variables auxiliares?

3) Escribir un programa que tome la entrada estándard, la codifique e imprima el resultado en salida estándard. La codificación deberá ser hecha caracter a caracter utilizando XOR y un código que se pasa como parámetro. ¿Qué modificaciones le tengo que hacer al programa para que decodifique? ¿Gano algo codificando más de una vez?. Probarlo codificando el fuente del programa y utilizando como código -98.

4) *Russian Peasant Algorithm.* La multiplicación de enteros positivos puede implementarse con sumas, ands y desplazamientos de bits usando las siguientes identidades:

$$a.b = \begin{cases} 0 & \text{si } b = 0 \\ a & \text{si } b = 1 \\ 2a.k & \text{si } b = 2k \\ 2a.k + a & \text{si } b = 2k + 1 \end{cases}$$

Usarlas para implementar una función:

`unsigned mult(unsigned a, unsigned b)`.

5) La arquitectura x86_64 restringe los enteros a 64 bits. ¿Qué sucede si ese rango no nos alcanza? Una solución es extender el rango utilizando más de un entero (en este caso enteros de 16 bits) para representar un valor. Así podemos pensar que

```
typedef struct { unsigned short n[16]; } nro;
```

representa el valor:

$$\begin{aligned} N \quad = \quad & nro.n[0] + \\ & nro.n[1] * 2^{sizeof(short)*8} + \\ & nro.n[2] * 2^{2*sizeof(short)*8} + \\ & ... + \\ & nro.n[15] * 2^{15*sizeof(short)*8} \end{aligned}$$

Podemos pensar en la estructura `nro` como un entero de 256 bits. Lamentablemente la arquitectura no soporta operaciones entre valores de este tipo por lo cual debemos realizarlas en software.

- Implemente funciones que comparen con 0 y con 1 y determinen paridad para valores de este tipo.
- Realice funciones que corran a izquierda y derecha los valores del tipo `nro`.
- Implemente la suma de valores del tipo `nro`.

Nota: En el repositorio de la materia hay una función para imprimir valores de este tipo. Esta función utiliza la librería GMP por lo cual deberá compilar el código agregando la opción `-lgmp`.

6) Implemente el Algoritmo del Campesino Ruso para los números anteriores.