Laboratorio 6- Programación Práctica 0

1. Diseñe e implemente un programa que solicite al usuario un número (en base 10) y una nueva base (en el intervalo [2,16]), e imprima la representación de ese número usando la base ingresada.
   Para coeficientes entre 10 y 15 utilice los caracteres entre 'A' y 'F'.
   Ejemplo:
   284 en base 16: 11C
   284 en base 4: 10130
   284 en base 8: 434
   284 en base 13: 18B
2. Modifique el programa anterior y encapsule el algoritmo en una función que reciba el valor y la nueva base, siendo la salida todavía realizada en el dispositivo estándar de salida.
3. Genere un archivo header con el prototipo de la función realizada en el ejercicio anterior y separe el main en otro archivo. Compile y corra nuevamente.
4. Realizar un programa que funcione como filtro de comentarios, definiendo como comentario a todo aquello colocado entre paréntesis (con posibilidad de comentarios anidados). Por ejemplo, si se ingresa: "Juan (mi mejor amigo) llega hoy de viaje" la salida sería "Juan llega hoy de viaje".
5. Implemente las siguientes funciones de manejo de bits:

```
// retorna la cantidad de bits en 1 presentes en la representacion de 'value'
int numOnes(int value);

// devuelve el valor 'value', con el bit 'bit' en 1
int setBit(int value, int bit);

// devuelve el valor 'value', con el bit 'bit' en 0
int resetBit(int value, int bit);

// chequea si el bit 'bit' esta en 1
int testBit(int value, int bit);
```

6. Implementar la función StrStr que reciba dos arreglos de caracteres y busque sobre el primero de ellos por la ocurrencia del segundo.

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

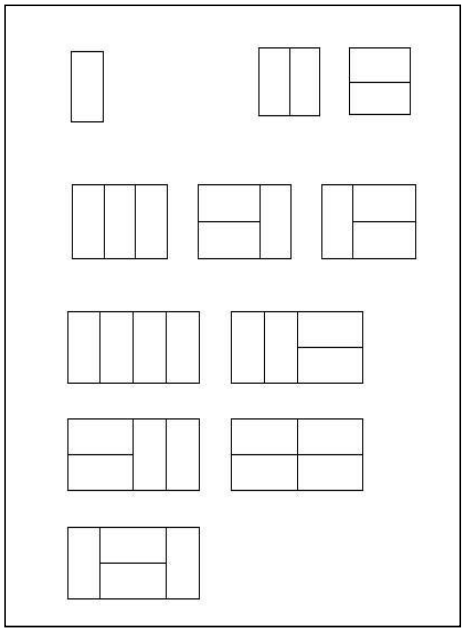
Como retorno, StrStr deberá retornar el índice del elemento sobre s1 donde comienza la ocurrencia de s2. Si s2 no aparece en s1, la función deberá retornar -1. Ejemplos:
`StrStr("JUAN ESTA CASADO CON MARIA", "ASADO")` retornará 11.
`StrStr("ABCDE", "BCE")` retornará -1.

7. Escribir una función que reciba como parámetro una cadena de caracteres que puede comenzar con espacios en blanco, y los elimine desplazando los caracteres útiles hacia la izquierda. (operación "left-trim").

   Prototipo: **`void left_trim (char phrase[]);`**

8. 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 >= B, el MCD entre A y B es equivalente al MCD entre B y el resto de la división entera entre A y B (MCD(A, B) == MCD(B, A%B)). (Hint: el MCD entre cualquier número X y cero es X).

9. Se desea averiguar de cuantas formas posibles se puede llenar un rectángulo de 2xn casilleros utilizando fichas de dominó (de 2x1). 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 2xn casilleros.

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

10. Validación de CBU: La Clave Bancaria Uniforme (CBU) es un código utilizado en Argentina por los bancos para la identificación de las cuentas de sus clientes. Mientras que cada banco identifica las cuentas con números internos, la CBU identifica una cuenta de manera única en todo el sistema financiero argentino. Está compuesta por 22 dígitos, separados en dos bloques (de 8 y 14 dígitos respectivamente). Por ejemplo:

28505909 40090418135201

En el primer bloque, los primeros 3 dígitos corresponden al Banco. Luego le sigue el primer "dígito verificador". Los siguientes 3 dígitos identifican la sucursal y por último el segundo dígito verificador.
El segundo bloque se compone de 13 dígitos para el número de cuenta y el tercer y último dígito verificador.
En el ejemplo dado sería:

Banco: **285** (B_0=2, B_1=8, B_2=5) DV_0 = 0

Sucursal: **590** (S_0=5, S_1=9, S_2=0) DV_1 = 9

Cuenta: **4009041813520** (C_0=4, C_1=0, C_2=0, …) DV_2 = 1

El procedimiento para validar una clave CBU es el siguiente:

- Validación del primer bloque: Se deben sumar los dígitos correspondientes al Banco, DV_0, y Sucursal, con los siguientes pesos: (7, 1, 3, 9, 7, 1, 3). En el ejemplo sería B_0*7 + B_1*1 + B_2*3 + DV_0*9 + S_0*7 + S_1*1 + S_2*3=81. El último dígito de esta suma se resta de 10 (En el ejemplo: DIF_1=10-1=9) y este valor debe coincidir con el DV_1 (En el ej. DV_1=9=DIF_1).
- Validación del segundo bloque: Sumar los dígitos del número de cuenta, con los siguientes pesos: (3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3). En el ejemplo sería (C_0*3 + C_1*9 + C_2*7 + C_3*1 + C_4*3 + C_5*9 + C_6*7 + C_7*1 + C_8*3 + C_9*9 + C_{10}*7 + C_{11}*1 + C_{12}*3=139). El último dígito de esta suma se resta de 10 (En el ejemplo: DIF_2=10-9=1) y este valor debe coincidir con el DV_2 (En el ej. DV_2=1=DIF_2).

Implemente una función que reciba una CBU en forma de string nativo, y lo verifique. Debe retornar verdadero o falso. Respete el siguiente prototipo:

```
int cbu_check(const char cbu[]);
```

Implemente un programa que solicite al usuario un número de CBU, lo verifique e imprima por pantalla si el número es válido o no. Implemente la función de forma que resulte sencillo cambiar los pesos que se le da a cada dígito.

11. Se desea implementar un functor **`Noiserer`**, que cuando es activado con un **`double`** como argumento retorne un valor que resulte de sumar ese argumento con una componente pseudoaleatoria. La componente pseudo-aleatoria estará limitada entre 2 umbrales porcentuales **`minVPercent`** y **`maxVPercent`** que serán indicados en la construcción del objeto functor. Verifique su funcionamiento utilizando el functor como función de transformación en la **`std::transform`** desde un **`vector<double> in`** a otro **`vector<doble> out`**, de tal forma que los elementos de **`out`** sean iguales a los de in, pero afectados con un **`Noiserer`**

```
class Noiserer {
    public:
```

```
        Noiserer (double minV_, double maxV_);
        double operator()(double v);
    private:
        double minV, maxV;
};
```

Claves públicas RSA

El sistema de claves públicas RSA permite la transmisión segura de datos. Está basado en la dificultad no-polinomial[1] de encontrar los factores primos de un número. Cada usuario debe generar una clave pública e y otra privada d a partir de un número n grande (e, d y n son números enteros).

Codificación y decodificación

Supongamos que Roberto quiere enviar un mensaje m a Alicia, donde m es un número entero. Para cifrarlo, simplemente calcula:

$$c = m^e \ \% \ n$$

($\%$ es la operación módulo, resto de la división entera) utilizando la clave pública e de Alicia y le envía el mensaje codificado c.

Alicia para decodificarlo calcula:

$$m = c^d \ \% \ n$$

utilizando su clave privada d, que solo ella conoce.

e, d y n deben ser obtenidos de manera que las operaciones que realizan Roberto y Alicia sean inversas una de la otra y que conociendo solo e y n sea muy difícil calcular d.

a) Implementar una función que calcule el exponente modular de un número:

$$x^{n_1} \ \% \ n_2$$

```
long exponente_modular(long x, long n1, long n2);
```

En general n_1 va a ser un número grande, por lo que va a ser necesario realizar las multiplicaciones de a una y utilizar que

$$x^s \ \% \ n_2 = (x * x^{s-1}) \ \% \ n_2 = ((x \ \% \ n_2) * (x^{s-1} \ \% \ n_2)) \ \% \ n_2$$

[1] Complejidad NP: la cantidad de operaciones necesarias crece exponencialmente con la cantidad de bits del número.

Ayuda: se puede verificar que esta expresión funciona adecuadamente utilizando la siguiente igualdad válida para cualquier número primo $p > 2$.

$$2^{p-1} \,\%\, p = 1,$$

por ejemplo: $2^6 \% 7 = (1 + 7 * 9) \% 7 = 1$.

b) Implementar la función cifra que reciba un **string** y retorne un **MensajeCifrado** cifrando carácter a carácter utilizando un e y n dados.

```
using MensajeCifrado = vector<long>;
MensajeCifrado cifra(const string &s, long e, long n);
```

c) Implementar una función descifra que reciba un **MensajeCifrado** y retorne un **string** con el mensaje descifrado a partir de un d y n dados.

```
string descifra(const MensajeCifrado &msg, long e, long n);
```

d) Utilice la función **descifra** para descifrar el siguiente **MensajeCifrado** utilizando $d = 1553$ y $n = 6767$.

```
MensajeCifrado mensaje { 4162, 5524, 789, 1445, 4537, 3698, 964, 1273, 4252, 1412,
                         3698, 6600, 5526, 5526, 2759, 2757, 2333};
```

Generación de claves

En las aplicaciones, n se calcula como el producto de dos números primos distintos obtenidos al azar.

e) Implementar una función que retorne un número primo al azar en el intervalo $[30: 200]$.

```
long rand_primo();
```

Ayuda: todos los números menores a 341 que satisfacen $2^{p-1} \,\%\, p = 1$ son primos, así que es posible utilizar esta igualdad para saber si un número es primo o no.

Utilizar esta función para obtener dos números primos distintos p y q.

La clave pública e tiene que ser un número coprimo con $r = (p-1) * (q-1)$. Se recuerda que dos números son coprimos si el máximo común divisor entre ambos es 1, esto es: $mcd(e, r) = 1$.

f) Implementar una función que retorne un número e en el intervalo $[17: r)$ cuidando que e y r sean coprimos.

```
long coprimo(long r);
```

Ayuda 1: utilizar la función creada en la práctica 6, ejercicio 2.
Ayuda 2: comenzar con 17 e ir aumentándolo de a 2 hasta que sea coprimo con r.

Para calcular la clave privada d hay que obtener el inverso módulo r de e, esto es resolver la ecuación:

$$(e * d) \% r = 1, \quad (1)$$

Para ello se puede utilizar el algoritmo de Euclides extendido (una extensión del algoritmo del ejercicio 2 de la práctica 6) que permite resolver ecuaciones del tipo:

$$a * x + b * y = mcd(a, b),$$

Obteniendo x e y que son enteros y $mcd(a, b)$ que es el máximo común divisor de a y b. Como para el algoritmo de Euclides:

- Si $b = 0$ la solución es trivial: $x = 1$, $y = 0$ y $mcd(a, b) = a$.
- Si $b \neq 0$ y tenemos la solución para b y $a\%b$:

$$mcd(b, a\%b) = b * x' + (a\%b) * y'$$

se puede demostrar que $x = y'$ e $y = x' - (a/b) * y'$ son una solución de:

$$mcd(a, b) = a * x + b * y.$$

g) Implementar la función:

```
Terna eulerext(long a, long b);
```

que dados a y b retorne $\{x, y, mcd(a, b)\}$ en una estructura

```
struct Terna {
    long x;
    long y;
    long mcd;
};
```

Ayuda: en pseudo-código es:

- Si $b = 0$ retornar la solución trivial $\{1,0, a\}$.
- Si $b \neq 0$:
  - Llamar a `eulerext(b, a % b)` para obtener x', y' y mcd.
  - Actualizar $x = y'$ e $y = x' - (a/b) * y'$.
  - Retornar $\{x, y, mcd\}$.

La ecuación (1) se puede escribir como:

$$e * d + q * r = mcd(e, r)$$

ya que $mcd(e, r) = 1$ por construcción.

En base a esto:

h) Implementar la función:

```
Claves genera_claves();
```

que retorne una estructura

```
struct Claves {
    long e;
    long d;
    long n;
};
```

En caso que d sea negativo, calcular $d = (d + r)\%r$. Verificar que $(e * d)\%r = 1$.

Ejemplo: para $p = 67$, $q = 101$, $n = 6767 = (67 * 101)$, se obtienen $r = 6600 = (66 * 100)$, $e = 17$ y $d = 1553$.

El interés de esta codificación es que si no se conocen p y q es muy difícil de calcular d.

i) Utilizar las claves generadas y las funciones implementadas para codificar y decodificar un **`string`**.