

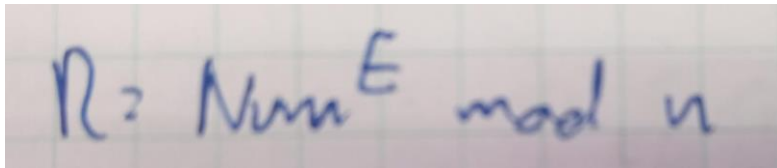
ALGEBRA ABSTRACTA
TERCER CONTROL

1. Escribir un programa que halle el residuo R de dividir un número Num elevado un exponente E módulo n

$$R = \text{PotenciaModulo}(\text{Num}, E, n)$$

Inserte el código del programa y en modo de comentario diga:

- a. (1 punto) Pasos relevantes que usa para hallar este residuo
- Basados en la siguiente fórmula:



A photograph of a piece of lined paper with the formula $R = \text{Num}^E \text{ mod } n$ written in blue ink.

- Con esto sacamos el exponente modulo n

```
// Función para calcular (base^exponente) % modulo de forma eficiente
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod; // Reducir la base si es mayor que el módulo

    while (exp > 0) {
        // Si el exponente es impar, multiplicamos el resultado por la base
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }

        // Exponente se reduce a la mitad y la base se eleva al cuadrado
        exp = exp >> 1;
        base = (base * base) % mod;
    }

    return result;
}
```

- Luego llamamos a la función principal

```

int main() {
    long long base, exponente, modulo;

    // Solicitar la entrada de base, exponente y módulo
    cout << "Introduce la base: ";
    cin >> base;
    cout << "Introduce el exponente: ";
    cin >> exponente;
    cout << "Introduce el modulo: ";
    cin >> modulo;

    // Verificar si el módulo es mayor que 1 para evitar errores
    if (modulo <= 1) {
        cout << "El modulo debe ser mayor que 1." << endl;
        return 1;
    }

    // Calcular base^exponente % modulo utilizando la función de exponenciación modular
    long long resultado = modExp(base, exponente, modulo);

    // Mostrar el resultado
    cout << "El resultado de " << base << "^" << exponente << " % " << modulo << " es: " << resultado << endl;

    return 0;
}

```

Donde solicitamos una base un exponente y un numero de módulo, para luego hacer uso de la fórmula y obtener el resultado

- b. (1 punto) Finalidad de cada una de las variables locales utilizadas
- En la función modExp, donde nos encargamos de todos recibe tres parámetros uno que es la base, otro que es el exponente y finalmente el número que recibe el modulo

```

// Función para calcular (base^exponente) % modulo de forma eficiente
long long modExp(long long base, long long exp, long long mod) {

```

Luego evaluamos si el exponente es mayor que 0 para operar con ello de ser caso contrario ósea con un exponente igual el resultado será 1

```

    long long result = 1;

```

Después evaluamos si el exponente es par para multiplicar el resultado por la base y finalmente reducir el exponente y hallar el modulo de la base al cuadrado devolviéndonos el resultado.

- c. (1 punto) Finalidad de cada una de las funciones invocadas en el proceso
- Aquí usamos únicamente dos funciones modExp y la función principal:
 - a. En modExp solicitamos los componentes de la formula y luego se encarga de calcular el resultado con lo antes mencionado en el punto anterior.
 - b. Finalmente usamos la función principal como formato de salida para la ejecución del programa
- d. (2 puntos) Ilustre el cálculo del residuo con Num=327, E=128, n=1425

```
Introduce la base: 327
Introduce el exponente: 128
Introduce el modulo: 1425
El resultado de 327^128 % 1425 es: 681

...Program finished with exit code 0
Press ENTER to exit console.
```

El resultado es 681.

2. Escribir un programa que genere una clave pública e y otra privada d a partir de dos números primos p, q

$$[e, d] = \text{GeneraClave}(p, q)$$

Inserte el código del programa y en modo de comentario diga:

- a. (1 punto) Pasos relevantes para hallar estas claves.

- Primero debeos solicitar los numero p y q

```
int main()
{
    long long p = 0;
    long long q = 0;
    long long phideN = phi(p,q);
    long long e = encontrar_e(phideN);
    long long d = inverso_modular(e,phideN);
}
```

- Luego debemos de hallar e que es el resultado de que en el mcd de ϕ de n y e salga 1.

```
1 #include <iostream>
2 using namespace std;
3
4 // Función para calcular el MCD (Máximo común divisor) de dos números
5 long long mcd(long long a, long long b) {
6     while (b != 0) {
7         long long temp = b;
8         b = a % b;
9         a = temp;
10    }
11    return a;
12 }
13
```

- Luego hallamos el inverso modular de e para hallar la clave privada d

```

29 // Algoritmo extendido de Euclides para encontrar el inverso de e módulo  $\phi(n)$ 
30 long long inverso_modular(long long e, long long phi) {
31     long long t = 0;
32     long long new_t = 1;
33     long long r = phi;
34     long long new_r = e;
35
36     while (new_r != 0) {
37         long long quotient = r / new_r;
38
39         // Intercambiamos los valores de t, new_t, r y new_r
40         long long temp_t = t;
41         t = new_t;
42         new_t = temp_t - quotient * new_t;
43
44         long long temp_r = r;
45         r = new_r;
46         new_r = temp_r - quotient * new_r;
47     }
48
49     if (r > 1) {
50         cout << "No existe inverso modular!" << endl;
51         return -1; // No tiene inverso
52     }
53
54     if (t < 0) {
55         t = t + phi; // Hacemos que el inverso sea positivo
56     }
57
58     return t;
59 }
60

```

b. (1 punto) Finalidad de cada una de las variables locales utilizadas

- En el caso del mcd almacenamos un a y un b que nos ayudaran a hacer las divisiones contantes hasta que ambos sean 1.

```

1  #include <iostream>
2  using namespace std;
3
4  // Función para calcular el MCD (Máximo común divisor) de dos números
5  long long mcd(long long a, long long b) {
6      while (b != 0) {
7          long long temp = b;
8          b = a % b;
9          a = temp;
10     }
11     return a;
12 }
13

```

- En el caso del inverso modular usamos los números e y phi de n para luego usar las ecuaciones de residuos y sacar el inverso

```

29 // Algoritmo extendido de Euclides para encontrar el inverso de e módulo  $\phi(n)$ 
30 long long inverso_modular(long long e, long long phi) {
31     long long t = 0;
32     long long new_t = 1;
33     long long r = phi;
34     long long new_r = e;
35
36     while (new_r != 0) {
37         long long quotient = r / new_r;
38
39         // Intercambiamos los valores de t, new_t, r y new_r
40         long long temp_t = t;
41         t = new_t;
42         new_t = temp_t - quotient * new_t;
43
44         long long temp_r = r;
45         r = new_r;
46         new_r = temp_r - quotient * new_r;
47     }
48
49     if (r > 1) {
50         cout << "No existe inverso modular!" << endl;
51         return -1; // No tiene inverso
52     }
53
54     if (t < 0) {
55         t = t + phi; // Hacemos que el inverso sea positivo
56     }
57
58     return t;
59 }
60

```

- luego en phi de n usamos los números p y q para multiplicarlos a cada uno reduciéndole una unidad.

```

//con esto hallamos el phi de n
long long phi(long long p, long long q)
{
    return (p-1)*(q-1);
}

```

- (1 punto) Finalidad de cada una de las funciones invocadas en el proceso
 - Como ya se mencionó, la función del inverso modular se encarga de almacenar e y phi de n para hallar el d que sería la clave privada.
 - En el mcd hallamos un máximo común divisor normal, pero en este paso lo usaremos para hallar la clave pública e
 - En la phi de n nos encargamos de hallar el valor phi de n para las otras dos funciones.
- (2 puntos) Ilustre el cálculo del residuo con p=19, q=51

```

Ingrese un numero primo p: 19
Ingrese otro numero primo q: 51
El numero phi de n es: 900
La clave pública E es: 7
La clave privada D es: 643

...Program finished with exit code 0
Press ENTER to exit console.

```

E = 7 y D = 643.

3. Escribir un programa que **cifre** un mensaje utilizando el algoritmo RSA

$C = \text{CifradoRsa}(\text{Mensaje}, \text{ClavePública})$

Donde:

- C es el mensaje cifrado.
- ClavePublica = (n, e);
- n es el producto de dos números primos
- e es el número coprimo con $\phi(n)$

Considere que, en el cifrado, cada carácter es un bloque.

Inserte el código del programa y en modo de comentario diga:

a. (1 punto) Pasos relevantes para realizar el cifrado

- Pues aquí seguimos la formula donde:
- $C = (\text{mensaje}^{\text{clave publica}}) \bmod n$
- Donde N es el numero producto de sus factores primos p y q.

```
include <iostream>
using namespace std;

// Función para realizar la exponenciación modular: (base^exp) % mod
long long exp_mod(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1) // Si exp es impar, multiplicamos el resultado por la base
            result = (result * base) % mod;
        exp = exp >> 1; // Dividimos exp entre 2
        base = (base * base) % mod; // Elevamos la base al cuadrado
    }
    return result;
}

int main() {
    long long n, e, d, mensaje, resultado;
    int opcion;

    cout << "Ingrese el valor de n (parte de la clave pública): ";
    cin >> n;
    cout << "Ingrese el valor de e (clave pública para cifrado): ";
    cin >> e;
    cout << "Ingrese el valor de d (clave privada para descifrado): ";
    cin >> d;

    cout << "Seleccione una opción:\n1. Cifrar\n2. Descifrar\nOpción: ";
    cin >> opcion;

    if (opcion == 1) { // Cifrar
        cout << "Ingrese el número a cifrar: ";
        cin >> mensaje;
        resultado = exp_mod(mensaje, e, n);
        cout << "Número cifrado: " << resultado << endl;
    }
    else if (opcion == 2) { // Descifrar
        cout << "Ingrese el número cifrado: ";
        cin >> mensaje;
        resultado = exp_mod(mensaje, d, n);
        cout << "Número descifrado: " << resultado << endl;
    }
    else {
        cout << "Opción no válida." << endl;
    }

    return 0;
}
```

b. (1 punto) Finalidad de cada una de las variables locales utilizadas

Pues en esta parte usamos el mismo algoritmo que la exponenciación modular ya que la fórmula es la misma salvo que usa únicamente, así que usamos las variables de una base, un exponente y un módulo. Eso lo único que hace esta parte ya que el desciframiento es lo mismo, la única diferencia son los datos de la fórmula.

```
#include <iostream>
using namespace std;

// Función para realizar la exponenciación modular: (base^exp) % mod
long long exp_mod(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1) // Si exp es impar, multiplicamos el resultado por la base
            result = (result * base) % mod;
        exp = exp >> 1; // Dividimos exp entre 2
        base = (base * base) % mod; // Elevamos la base al cuadrado
    }
    return result;
}
```

- c. (1 punto) Finalidad de cada una de las funciones invocadas en el proceso

Pues ya está explicado lo de exponenciación modular, solo faltaría la función principal donde pedimos el número n , e de la clave pública y d de la privada, pedir que se elija si cifrar o descifrar el mensaje y luego determinar un formato de salida.

```
int main() {
    long long n, e, d, mensaje, resultado;
    int opcion;

    cout << "Ingrese el valor de n (parte de la clave pública): ";
    cin >> n;
    cout << "Ingrese el valor de e (clave pública para cifrado): ";
    cin >> e;
    cout << "Ingrese el valor de d (clave privada para descifrado): ";
    cin >> d;

    cout << "Seleccione una opción:\n1. Cifrar\n2. Descifrar\nOpción: ";
    cin >> opcion;

    if (opcion == 1) { // Cifrar
        cout << "Ingrese el número a cifrar: ";
        cin >> mensaje;
        resultado = exp_mod(mensaje, e, n);
        cout << "Número cifrado: " << resultado << endl;
    }
    else if (opcion == 2) { // Descifrar
        cout << "Ingrese el número cifrado: ";
        cin >> mensaje;
        resultado = exp_mod(mensaje, d, n);
        cout << "Número descifrado: " << resultado << endl;
    }
    else {
        cout << "Opción no válida." << endl;
    }

    return 0;
}
```

- d. (2 puntos) Ilustre el cifrado del mensaje "HOLA MUNDO"

Tengo tres programas uno para las claves privadas, otro que se encarga del RSA, y el mago del ejercicio en encargado de convertir números a letra en ASCII

-primero convertirnos HOLA MUNDO en números

```
-----  
NUMEROS A LETRAS Y AL REVES  
-----  
Escoge una opcion:  
1.(letras a numeros ASCII)  
2.(numeros ASCII a letras)  
1  
Usted ha elegido convertir letras a numeros ASCII  
Ingrese una frase: HOLA MUNDO  
Su frase en numeros ASCII es: 72 79 76 65 32 77 85 78 68 79
```

HOLA MUNDO en números es: 72 79 76 65 77 85 78 68 79, el 32 es el espacio así que no lo contamos.

- Usamos el p y q del problema 3 donde p es 19 y q es 51, por lo tanto, n es el producto de estos dos números

$$19 \times 51 =$$
$$969$$

- 969 será nuestro n

```
Ingrese un numero primo p: 19  
Ingrese otro numero primo q: 51  
El numero phi de n es: 900  
La clave p-blica E es: 7  
La clave privada D es: 643
```

- Ahora sacamos la phi de $n = (p - 1) * (q - 1) = 900$
- Donde E de la clave publica es: 7
- D de la clave privada es: 643
- Para cifrar reemplazamos en la fórmula
- (72 79 76 65 77 85 78 68 79), cada valor elevado a 7 con modulo 969
- Entonces
- 72 79 76 65 77 85 78 68 79 (HOLA MUNDO) cifrado es:
- 336 496 304 635 (HOLA) cifrado
- 818 289 90 68 496 (MUNDO) cifrado
- 336 496 304 635 818 289 90 68 496, respuesta
- Prueba de cifrado de RSA por unidades, es decir letra por letra


```

Ingrese el valor de n (parte de la clave p.blica): 969
Ingrese el valor de e (clave p.blica para cifrado): 7
Ingrese el valor de d (clave privada para descifrado): 643
Seleccione una opci n:
1. Cifrar
2. Descifrar
Opci n: 1
Ingrese el n mero a cifrar: 79
N mero cifrado: 496

```

-
- Conversi n de la letra O = 79 en ASCII, cifrado con RSA = 496

4. Escriba un programa que **descifre** un cifrado utilizando el algoritmo

RSA Mensaje=DescifradoRsa(C, ClavePrivada)

Donde:

- Mensaje es el mensaje descifrado
- ClavePrivada = (n, d)
- n=es el producto de dos n meros primos
- d es el inverso de e (clave privada)

Considere que, en el cifrado, cada car cter es un bloque

Inserte el c digo del programa y en modo de comentario diga:

- a. (1 punto) Pasos para realizar el cifrado
 - Igualmente seguimos una formula por exponenciaci n modular:
 - Mensaje = (cifrado ^ clave privada) modulo n
 - Entonces usamos lo mismo
- b. (1 punto) Finalidad de cada una de las variables locales utilizadas
 - Pues realmente es igual al ejercicio anterior con las mismas variables, si que la funci n es la misma
- c. (1 punto) Finalidad de cada una de las funciones invocadas en el proceso
 - Igual que en el ejercicio 4.
- d. (2 puntos) Ilustre el descifrado del mensaje "24 58 125 130 254"
 - Tenemos (24 58 125 130 254) mensaje cifrado, elevamos cada elemento con la clave privada (p= 11 y q=13), por lo que n = 143, e=7 y d=103
 - Reemplazando tenemos:
 - Mensaje descifrado = (24 58 125 130 254) ^ 1 modulo 6
 - 41 137 31 91 45 que seria el mensaje decifrado, usando los programas no tiene equivalente en letras

NUMEROS A LETRAS Y AL REVES

Escoge una opcion:

- 1.(letras a numeros ASCII)
- 2.(numeros ASCII a letras)

2

Usted ha elegido convertir numeros ASCII a letras

Ingrese una serie de numeros ASCII separados por espacios: 41 137 31 91 45

La frase convertida es: @ë▼[-