

# Estructura de Datos y Algoritmia

RECOPIACIÓN DE EJERCICIOS EN C

# Índice general

<b>1. Aviso legal</b>	<b>4</b>
<b>2. Agradecimiento</b>	<b>5</b>
<b>3. Prólogo</b>	<b>6</b>
<b>4. Complejidad computacional</b>	<b>7</b>
Algoritmos . . . . .	7
<b>5. Camino de piedras</b>	<b>10</b>
Entrada . . . . .	10
Salida . . . . .	10
Ejemplos . . . . .	10
Solución . . . . .	12
<b>6. Festín de chocolate</b>	<b>14</b>
Entrada . . . . .	14
Salida . . . . .	14
Ejemplos . . . . .	14
Solución . . . . .	16
<b>7. Salva al prisionero</b>	<b>17</b>
Entrada . . . . .	17
Salida . . . . .	17
Ejemplos . . . . .	18
Solución . . . . .	19
<b>8. Matriz caracol</b>	<b>21</b>
Entrada . . . . .	21
Salida . . . . .	21
Ejemplos . . . . .	21
Solución . . . . .	22
<b>9. Grafo dirigido</b>	<b>25</b>
Entrada . . . . .	25

Salida . . . . .	25
Ejemplos . . . . .	25
Solución . . . . .	27
<b>10.Triángulo de asteriscos</b>	<b>29</b>
Entrada . . . . .	29
Salida . . . . .	29
Ejemplos . . . . .	29
Solución . . . . .	31
<b>11.Lectura de archivos</b>	<b>32</b>
Entrada . . . . .	32
Salida . . . . .	32
Ejemplos . . . . .	32
Solución . . . . .	33
<b>12.Recursividad</b>	<b>36</b>
Entrada . . . . .	36
Salida . . . . .	36
Ejemplos . . . . .	37
Solución . . . . .	38
<b>13.Triangulo de palabras</b>	<b>40</b>
Entrada . . . . .	40
Salida . . . . .	40
Ejemplos . . . . .	41
<b>14.Separar mayúsculas y minúsculas</b>	<b>43</b>
Entrada . . . . .	43
Salida . . . . .	43
Ejemplos . . . . .	44
Solución . . . . .	44
<b>15.Verificar Suma</b>	<b>46</b>
Entrada . . . . .	46
Salida . . . . .	46
Ejemplos . . . . .	46
Solución . . . . .	47
<b>16.Cifrado numérico</b>	<b>49</b>
Entrada . . . . .	50
Salida . . . . .	50
Ejemplos . . . . .	50
Solución . . . . .	51
<b>17.Búsqueda en anchura: alcance más corto</b>	<b>55</b>
Restricciones . . . . .	55

Entrada . . . . .	55
Salida . . . . .	56
Ejemplos . . . . .	56
Solución . . . . .	57
<b>18.Palabra rangoli</b>	<b>59</b>
Entrada . . . . .	59
Salida . . . . .	60
Ejemplos . . . . .	60
Solución . . . . .	61
<b>19.Sherlock Holmes y la Bestia</b>	<b>63</b>
Entrada . . . . .	63
Salida . . . . .	64
Ejemplos . . . . .	64
Solución . . . . .	64
<b>20.Mensajes cifrados</b>	<b>66</b>
Entrada . . . . .	66
Salida . . . . .	66
Ejemplos . . . . .	67
Solución . . . . .	67
<b>21.Criba de Erastótenes</b>	<b>69</b>
Entrada . . . . .	69
Salida . . . . .	69
Ejemplos . . . . .	70
Solución . . . . .	70
<b>22.Cifrado de vigenere</b>	<b>72</b>
Entrada . . . . .	72
Salida . . . . .	72
Ejemplos . . . . .	73
Solución . . . . .	74
<b>23.Cuys</b>	<b>75</b>
Entrada . . . . .	75
Salida . . . . .	75
Ejemplos . . . . .	76
Solución 1 . . . . .	76
Solución 2 . . . . .	77

# Capítulo 1

## Aviso legal

**Estructura de Datos y Algoritmia: recopilación de ejercicios en c.**

Por: Carlos Cevallos, Cristhian Hernández, David Guillermo, Doménica Lasso, Franklin Lara, Santos Gallegos.

Este libro se distribuye de manera gratuita con fines educativos.

Usted es libre de:

- Copiar, distribuir de manera **gratuita** la obra.
- Realizar derivados (reconociendo y dando crédito a los autores originales).

**Copyright (c) 2016**

## Capítulo 2

# Agradecimiento

*El aprendizaje es la facultad del ser humano para salir de la ignorancia, los conocimientos adquiridos cada día y la práctica de cada uno de estos es lo que nos lleva al éxito.*

*Agradecemos de manera especial a la Ingeniera Malhena Sánchez quien nos impartió sus conocimientos y experiencias, permitiéndonos e incentivándonos a realizar este proyecto para poder dejar huellas de los conocimientos adquiridos en las aulas de clase.*

*Agradecemos a demás a cada una de las personas que de manera directa o indirecta han colaborado para poder realizar este documento.*

## Capítulo 3

# Prólogo

*La importancia de las ciencias de la computación en la actualidad es muy notable, lo podemos ver en funcionamiento desde una simple búsqueda en google hasta en una transacción bancaria. La rapidéz en que se relizan estas operaciones es gracias a la implementación de varios algoritmos en conjunto con varias estrucuras de datos.*

*En el siguiente documento se presenta una serie de ejercicios con varios niveles de dificultad que le permitirán desarrollar sus habilidaddes de resolución de problemas, además se plantea las respectivas soluciones más óptimas para cada ejercicio.*

## Capítulo 4

# Complejidad computacional

***Temas:** complejidad computacional, análisis de algoritmos, notación O-grande*

Calcular la complejidad computacional con la notación O-grande de los siguientes algoritmos.

### Algoritmos

#### A

```
int n;  
int suma = 0;  
printf("n: ");  
scanf("%d", &n);  
  
for (int i = n; i > 0; i /= 2) {  
    for (int j = 0; j < i; j++) {  
        suma++;  
    }  
}
```

#### Solución A

$O(\ln(n))$

---



## B

```
int n;  
int suma = 0;  
printf("n: ");  
scanf("%d", &n);  
  
for (int i = 1; i < n; i *= 2) {  
    for (int j = 0; j < i; j++) {  
        suma++;  
    }  
}
```

### Solución B

$O(\ln(n))$

---

## C

```
int k = 0;  
int s = 0;  
while (k < n) {  
    s += d[k];  
    if (k == 0)  
        k = 2;  
    else  
        k *= 2;  
}
```

### Solución C

$O(\ln(n))$

---

## D

```
int n;  
int suma = 0;  
printf("n: ");  
scanf("%d", &n);
```

```
for (int i = 0; i < n; i++) {  
    for (int j = i + 1; j < n; j++) {  
        for (int k = j + 1; k < n; k++) {  
            suma++;  
        }  
    }  
}
```

### Solución D

$O(n^3)$

## Capítulo 5

# Camino de piedras

*Temas: bucles, matemáticas*

Carla se encuentra con un camino de piedras marcadas con números. Empieza a seguir el camino, y nota que la diferencia entre dos piedras consecutivas es **a** o **b**. Cuenta la leyenda que hay un tesoro al final del camino, si Carla adivina el valor de la última piedra, el tesoro será suyo. Dado el número de la primera piedra es 0 (cero), encuentra todos los posibles valores de la última piedra.

### Entrada

La primer línea contiene **n**, el número de piedras. La segunda línea contiene **a**, y la tercera contiene **b**.

### Salida

Una lista de todos los valores posibles de las últimas piedras separadas por un espacio.

---

### Ejemplos

#### Entrada 0

3

1

2

### Salida 0

2 3 4

### Explicación

Todas las posibles series de piedras son:

0, 1, 2

0, 1, 3

0, 2, 3

0, 2, 4

Entonces, los posibles valores de la última piedra son:

2, 3, 4

### Entrada 1

6

4

8

### Salida 1

20 24 28 32 36 40

### Entrada 2

11

3

10

## Salida 2

30 37 44 51 58 65 72 79 86 93 100

## Entrada 3

100

1

1

## Salida 3

99

---

## Solución

```
#include <stdio.h>

int main() {
    int numero_piedras;
    int numero_a, numero_b;
    int min_ultima_piedra, max_ultima_piedra;

    printf("Numero de piedras: "); scanf("%d", &numero_piedras);
    printf("Numero a: "); scanf("%d", &numero_a);
    printf("Numero b: "); scanf("%d", &numero_b);

    min_ultima_piedra = numero_a*(numero_piedras - 1);
    max_ultima_piedra = numero_b*(numero_piedras - 1);

    if(numero_a == numero_b){
        printf("%d ", min_ultima_piedra);
    }else {
        for ( i = min_ultima_piedra;
            i <= max_ultima_piedra;
            i += (numero_b - numero_a) {

            printf("%d ", i);

        }
    }
}
```

```
    printf("\n");  
    return 0;  
}
```

## Capítulo 6

# Festín de chocolate

*Temas: bucles*

Bob ama el chocolate, y se dirige a la tienda con  $\$N$ . El precio de cada chocolate es  $\$C$ . La tienda ofrece un descuento: por cada  $M$  envolturas, recibes un chocolate gratis. ¿Cuántos chocolates puede obtener Bob?

### Entrada

La primera línea contiene un entero  $N$ , la cantidad de dinero que tiene Bob. La segunda línea contiene un entero  $C$ , el precio de cada chocolate. Y la última línea contiene un entero  $M$ , el número de envolturas por las cuales puede obtener un chocolate gratis.

### Salida

Mostrar el número total de chocolates que puede obtener Bob.

---

### Ejemplos

#### Entrada 0

6  
2

2

### Salida 0

5

### Explicación

Con \$6 Bob compra 3 chocolates, por lo tanto obtiene 3 envolturas. Ocupa 2 envolturas para reclamar un chocolate gratis. Ahora Bob tiene un total de 2 envolturas, las cuales usa para obtener otro chocolate gratis. En total Bob obtuvo 5 chocolates.

### Entrada 1

37

3

7

### Salida 1

13

### Entrada 2

369

96

33

### Salida 2

3



### Entrada 3

44112  
389  
22785

### Salida 3

1

---

### Solución

```
#include <stdio.h>
#include <stdbool.h>

int main() {
    int dinero_total;
    int valor_por_chocolate;
    int envolturas_para_promocion;

    printf("Dinero total:");
    scanf("%d", &dinero_total);

    printf("Valor de cada chocolate:");
    scanf("%d", &valor_por_chocolate);

    printf("Numero de envolturas para promocion:");
    scanf("%d", &envolturas_para_promocion);

    int chocolates_obtenidos = dinero_total/valor_por_chocolate;
    int envolturas = chocolates_obtenidos;

    while(envolturas >= envolturas_para_promocion){
        chocolates_obtenidos++;
        envolturas = envolturas - envolturas_para_promocion + 1;
    }

    printf("Numero de chocolates obtenidos: %d\n", chocolates_obtenidos);

    return 0;
}
```

## Capítulo 7

# Salva al prisionero

*Temas: aritmética modular, matemáticas*

En una cárcel hay  $N$  prisioneros, y cada uno tiene un identificador único  $S$ , en el rango de  $1$  a  $N$ . Hay  $M$  caramelos que deben ser entregados a cada prisionero. Dado que no siempre el número de caramelos es igual al de prisioneros, se decide sentar a los prisioneros en un círculo, ordenados por su id. Y se distribuyen los caramelos consecutivamente empezando por un id  $S$  aleatorio hasta que se terminen los caramelos. Pero hay una trampa, ¡el último caramelo está envenenado! ¿Puedes encontrar el id del prisionero que reciba el último caramelo, y así ser advertido?.

### Entrada

La primera línea contiene un entero  $N$ , el número de prisioneros. La segunda línea contiene un entero  $M$ , el número de caramelos. Y la tercer línea contiene un entero  $S$ , el id del prisionero.

### Salida

Mostrar el id del prisionero que recibirá el caramelo envenenado.

---

## Ejemplos

### Entrada 0

5  
4  
3

### Salida 0

1

### Explicación

Hay 5 prisioneros y 4 caramelos. Empezamos a repartir el primer caramelo al prisionero de id 3, hasta llegar al prisionero de id 5 se ha repartido 3 caramelos. Se entrega el cuarto caramelo al prisionero de id 1. Entonces, el último caramelo lo recibe el prisionero de id 1.

### Entrada 1

54  
111  
6

### Salida 1

8

### Entrada 2

499999999  
999999998  
2

## Salida 2

1

## Entrada 3

287704511

903048419

110994298

## Salida 3

150929183

---

## Solución

```
#include <stdio.h>

int main() {
    int num_prisioneros;
    int num_caramelos;
    int id_prisionero;

    printf("Numero de prisioneros: ");
    scanf("%d", &num_prisioneros);

    printf("Numero de caramelos: ");
    scanf("%d", &num_caramelos);

    printf("ID de prisionero: ");
    scanf("%d", &id_prisionero);

    int id_ultimo_prisionero;

    // Se resta uno para empezar a repartir desde el prisionero actual.
    id_ultimo_prisionero = (num_caramelos + id_prisionero - 1) % num_prisioneros;

    if(id_ultimo_prisionero == 0){
        id_ultimo_prisionero = num_prisioneros;
    }
```

```
printf("ID del ultimo prisionero: %d\n", id_ultimo_prisionero);  
  
return 0;  
}
```

## Capítulo 8

# Matriz caracol

*Temas: bucles, matrices*

Se pide crear una matriz caracol, que consiste en ingresar el el número de filas y el número de columnas, y el programa se encargara de llenar automaticamente la matriz, se llenara de izquierda a derecha, despues de arriba hacia abajo, luego de derecha a izquierda y finalmente de abajo hacia arriba.

## Entrada

La primer línea contiene **n**, el número de filas. La segunda línea contiene **m**, el número de columnas.

## Salida

La matriz caracol de tamaño **n x m**

---

## Ejemplos

### Entrada 0

3  
4

## Salida 0

```
| 1|| 2|| 3|| 4|
| 10|| 11|| 12|| 5|
| 9|| 8|| 7|| 6|
```

## Entrada 1

```
5
5
```

## Salida 1

```
| 1|| 2|| 3|| 4|| 5|
| 16|| 17|| 18|| 19|| 6|
| 15|| 24|| 25|| 20|| 7|
| 14|| 23|| 22|| 21|| 8|
| 13|| 12|| 11|| 10|| 9|
```

## Solución

```
#include <stdio.h>

#include <stdio.h>
#include <stdlib.h>

int n = 0,m = 0;
void generarMatriz(int n);

int main(){
    printf("Inserte las filas de la matriz: ");
    scanf("%d",&n);
    printf("Inserte las columnas matriz: ");
    scanf("%d",&m);
    generarMatriz(n);
    return 0;
}

void generarMatriz(n){
    int i = 0,j = 0, k = n,l = m, num_act = 0;
    int tamanoTotal = n*m,cont = 0;
```

```

int matrizCaracol[n][m];

for(j = 0; j<m ; j++ ){
    num_act += 1;
    matrizCaracol[0][j] = num_act;
}
j -= 1;

while(num_act<tamanoTotal){
    k -= 1;
    cont = 0;
    while((k > cont) && (l>0)){
        num_act += 1;
        i += 1;
        matrizCaracol[i][j] = num_act;
        cont += 1;
    }

    l -= 1;
    cont = 0;
    while((l > cont) && (k>0)){
        num_act += 1;
        j -= 1;
        matrizCaracol[i][j] = num_act;
        cont += 1;
    }

    k -= 1;
    cont = 0;
    while((k > cont) && (l>0)){
        num_act += 1;
        i -= 1;
        matrizCaracol[i][j] = num_act;
        cont += 1;
    }

    l -= 1;
    cont = 0;
    while((l > cont) && (k>0)){
        num_act += 1;
        j += 1;
        matrizCaracol[i][j] = num_act;
        cont += 1;
    }
}

```



```

for(i = 0; i<n ; i++ ){
    for(j = 0; j<m ; j++ ){
        if(matrizCaracol[i][j] < 10)
            printf("| %d|",matrizCaracol[i][j]);
        else if (matrizCaracol[i][j] < 100)
            printf("| %d|",matrizCaracol[i][j]);
        else
            printf("|%d|",matrizCaracol[i][j]);
    }
    printf("\n");
}
}

```

## Capítulo 9

# Grafo dirigido

*Temas: grafos*

Ingresa el número de nodos de la matriz, luego crear una matriz de adyacencia recibiendo los pesos de la matriz, y finalmente verificar si el grafo es dirigido o no.

### Entrada

La primer línea contiene **n**, el número de nodos. Las siguientes líneas contienen los pesos de la matriz adyacente.

### Salida

Texto mostrando si la matriz es o no dirigida.

---

### Ejemplos

#### Entrada 0

2

Pesos de la Matriz de Adyacencia

1  
2  
1  
2

### Salida 0

El grafo es dirigido.

### Explicación

Un grafo no es dirigido cuando la matriz de adyacencia es simétrica, en este caso la matriz no es simétrica, por ende no es dirigida.

### Entrada 1

3  
1  
2  
3  
2  
2  
2  
3  
2  
1

### Salida 1

El grafo no es dirigido.

---

## Solución

```
#include <stdlib.h>
#include <stdio.h>

int numero_nodos=0;
int matriz_adyacencia[5][5];

void ver_dirigido(){
    int i,j,cont=0;
    for(i=0;i<numero_nodos;i++){
        for(j=0;j<numero_nodos;j++){
            if(matriz_adyacencia[i][j]!=0){
                if(matriz_adyacencia[i][j]!=matriz_adyacencia[j][i]){
                    printf("\nEl grafo es dirigido.\n");
                    return;
                }
            }
        }
    }
    printf("\nEl grafo no es dirigido.\n");
}

void leer_pesos(){
    int i,j;
    for(i=0;i<numero_nodos;i++){
        for(j=0;j<numero_nodos;j++){
            printf("\nIngrese el peso entre %d->%d: ",i+1,j+1);
            scanf("%d",&matriz_adyacencia[i][j]);
        }
    }
}

void mostrar_matriz_pesos(){
    int i,j;
    printf("\nMatriz de Adyacencia\n\n");
    for(i=0;i<numero_nodos;i++){
        for(j=0;j<numero_nodos;j++){
            printf("|%3d|",matriz_adyacencia[i][j]);
        }
        printf("\n");
    }
}
```

```

int main(){
    printf("Ingrese el numero de nodos:");
    scanf("%d",&numero_nodos);
    while(numero_nodos<1 || numero_nodos>4){
        printf("\nIngrese numero de nodos <5 y >1:\n");
        scanf("%d",&numero_nodos);
    }
    leer_pesos();
    mostrar_matriz_pesos();
    ver_dirgido();

    return 0;
}

```

## Capítulo 10

# Triángulo de asteriscos

*Temas: recursividad*

Se desea formar un triángulo de altura **h**, con una base de **(2x8)-1**. Este triángulo debe ser formado usando funciones recursivas.

### Entrada

La primer línea contiene **h**, la altura del triángulo.

### Salida

Triángulo de altura **h**.

---

### Ejemplos

#### Entrada 0

3

#### Salida 0

\*

\*\*  
 \*\*\*  
 \*\*  
 \*

## Explicación

Si el triangulo es de altura 3 primero se va a imprimir un asterisco, luego 2 asteriscos, hasta llegar a los 3 asteriscos, una vez realizado esto, el numero de asteriscos a imprimirse se reducira hasta que sea 0.

## Entrada 1

10

## Salida 1

\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*\*  
\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*

```
***  
**  
*
```

---

## Solución

```
#include <stdlib.h>  
#include <stdio.h>  
  
void mostrar(int n){  
    if(n == 0){  
        printf("\n");  
        return;  
    } else {  
        printf("*");  
        mostrar(n-1);  
    }  
}  
  
void triangulo(int n,int h){  
    if (h == 0) {  
        return;  
    } else {  
        mostrar(n-h);  
        triangulo(n,h-1);  
        mostrar(n-h+1);  
    }  
}  
  
int main(){  
    int h;  
    printf("Ingrese la altura del triangulo: ");  
    scanf("%d",&h);  
    triangulo(h,h);  
  
    return 0;  
}
```



## Capítulo 11

# Lectura de archivos

***Temas:** archivos, chars*

Leer un archivo .txt y contabilizar el número de cada vocal que contiene el archivo sin importar si son mayúsculas o minúsculas. El usuario deberá elegir el archivo a contabilizar las vocales. Se debe pedir por teclado el nombre del archivo o la dirección donde se encuentre este.

### Entrada

Ingresar el nombre del archivo.

### Salida

Una vocal por línea junto con el número de ocurrencias.

---

### Ejemplos

#### Entrada 0

Ingrese el nombre del archivo: archivo.txt

**archivo.txt**

A I O U

I

ai

i

I

## Salida 0

a=2

e=0

i=5

o=1

u=1

## Explicación

Se obtiene la salida del número de vocales que hay en cada archivo que desea conocer el usuario, se debe contabilizar tanto vocales mayúsculas como minúsculas. La salida es el resultado de número de vocales encontradas en el archivo.

---

## Solución

```
# include <stdio.h>
# include <stdlib.h>

int a=0,e=0,i=0,o=0,u=0;

int existe(char nombre[200]){
    FILE* archivo;
    archivo=fopen(nombre,"r");
    if(archivo==NULL){
        printf("\nEl archivo no existe");
        fclose(archivo);
        return 0;
    }
```

```

    }
    else{
        fclose(archivo);
        return 1;
    }
}

void contarVocales(char nombre[200] ){
    FILE* archivo;
    archivo=fopen(nombre,"r");
    char letra;
    while (!feof(archivo)) {
        letra=fgetc(archivo);
        letra=tolower(letra);
        switch(letra){
            case 'a':
                a+=1;
                break;
            case 'e':
                e+=1;
                break;
            case 'i':
                i+=1;
                break;
            case 'o':
                o+=1;
                break;
            case 'u':
                u+=1;
                break;
            default:
                break;
        }
    }
}

int main(){
    char numero[100], archivo[200]="";
    int comprobar;
    printf("\nIngrese la ruta del archivo: \n");
    gets(archivo);
    comprobar= existe(archivo);
    if(comprobar==1){
        contarVocales(archivo);
        printf("Archivo Contiene: \na=%d\ne=%d\ni=%d\no=%d\nu=%d",a,e,i,o,u);
    }
}

```

}

## Capítulo 12

# Recursividad

**Temas:** Recursividad, Matrices

Se debe realizar un programa recursivo que pida el numero de la matriz que será cuadrada y un numero que sera la basa para los elementos de la matriz que sera llenada recursivamente.

```
Ingrese numero de la matriz: 3
Ingrese numero de base: 2
  1  1  1
  1  2  2
  1  2  4
```

## Entrada

Número de la matriz y número de la base.

## Salida

La matriz resultante.

---

## Ejemplos

### Entrada 0

4  
2

### Salida 0

1 1 1 1  
1 2 2 2  
1 2 4 4  
1 2 4 8

### Explicación

La matriz es cuadra cada fila y columna se llenara de forma resursiva, los elementos serán la base ingresada al exponente segun el numero de la fila y columna (exponente=fila=columna), se ira llenando dependiendo de la fila y de la columna, la fila uno y columna uno tendrá la base elevada el exponente 0, la fila 1 y columna 1 omitiendo el primer elemento de la fila y columna 1 tendrá la base elevada al numero 1 y asi hasta llegar a la fila y columna n, teninedo en cuenta que se debe omitir llenar los elemetnos de las filas y columnas ya antes llenadas.

### Entrada 1

5  
3

### Salida 1

1 1 1 1 1  
1 3 3 3 3  
1 3 9 9 9  
1 3 9 27 27

1 3 9 27 81

---

## Solución

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define tamaño 50

int matriz[tamaño][tamaño];
int recorrido(int,int,int,int,int,int);
int llenado(int,int,int,int,int,int);
void imprime(int);

int main(){
    int nmatriz,base;
    int exponente=1, numero=0,filas=0,columna=0;
    printf(" Ingrese numero de la matriz: ");
    scanf("%d",&nmatriz);
    printf("\n Ingrese numero de base: ");
    scanf("%d",&base);
    recorrido(numero,exponente,nmatriz,base,filas,columna);
    imprime(nmatriz);
    return 0;
}

int recorrido(
    int numero,
    int exponente,
    int nmatriz,
    int base,
    int filas,
    int columna
){
    if(numero==nmatriz)
        return 1;
    else
        llenado(numero,exponente,nmatriz,base,filas,columna);
}
```

```

int llenado(
    int numero,
    int exponente,
    int nmatriz,
    int base,
    int fila,
    int columna
){

    if(columna==nmatriz){
        numero+=1;
        fila+=1;
        columna=fila;
        exponente=exponente*base;
        return recorrido(numero,exponente,nmatriz,base,fila,columna);
    } else {
        matriz[fila][columna]=exponente;
        matriz[columna][fila]=exponente;
        columna+=1;
        return llenado(numero,exponente,nmatriz,base,fila,columna);
    }

}

void imprime(int nmatriz){
    int i,j;
    for(i=0;i<nmatriz;i++){
        for(j=0;j<nmatriz;j++){
            printf("%5d",matriz[i][j]);
        }
        printf("\n\n");
    }
}

```



## Capítulo 13

# Triangulo de palabras

*Temas: strings, bucles, condicionales*

Realizar un programa que reciba una palabra o frase y que se construya una triangulo con la palabra, pudiendo leerse la palabra por cada lado del triangulo.

```
Ingrese la frase: hola mundo
      h
     o o
    l  l
   a  a
  m    m
 u    u
n    n
d    d
odnum alohola mundo
```

## Entrada

Ingreso de la palabra o frase

## Salida

Debera observarse la palabra en forma de triangulo

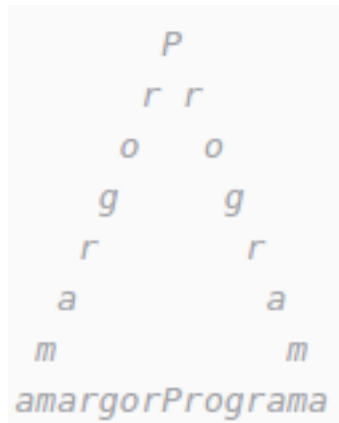
---

## Ejemplos

### Entrada 0

Programa

### Salida 0



```
      P
    r  r
  o    o
 g      g
r        r
a          a
m            m
amargorPrograma
```

### Explicación

Se observa que se forma un triangulo isocetes donde en los lados se lee la palabra ingresada y el la base debe salir como un espejo de la palabra desde la última letra hasta la primero y luego de la primera hasta la ultima letra como un espejo.

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

void espacios(int numero){
    int i=0;
    for(i=0;i<numero;i++){
        printf(" ");
    }
}

void triangulo(char frase[100]){
    int j,cantidad=0,distancia;
    cantidad=strlen(frase);
```

```

    distancia=cantidad-1;
    for(j=0;j<cantidad-1;j++){
        espacios(distancia);
        printf("%c",frase[j]);
        espacios(1+((j-1)*2));
        if(j!=0){
            printf("%c",frase[j]);
        }
        distancia-=1;
        printf("\n");
    }
    for(j=cantidad-1;j>0;j--){
        printf("%c",frase[j]);
    }
    printf("%s",frase);
}

int main (){
    char frase[100];
    int i,j, cantidad=0;
    printf("Ingrese la frase: ");
    gets(frase);
    cantidad=strlen(frase);
    triangulo(frase);
}

```

## Capítulo 14

# Separar mayúsculas y minúsculas

*Tema: bucles*

Se pide desarrollar un algoritmo que solicite una frase por pantalla que contenga tanto letras mayúsculas como minúsculas. Una vez aceptada la frase se ha de visualizar las letras mayúsculas y minúsculas por separado almacenando también las cadenas generadas en variables separadas, como ejemplo si se teclea HOla MunDO, la frase en minúsculas tendrá que contener las letras “laun” mientras que la variable que contiene las letras en mayúsculas ha de contener y visualizar “HOMDO”.

### Entrada

Se recibirá la frase por pantalla **Frase:.**

### Salida

Mensaje que indica la frase original, la cadena de mayúsculas y la de minúsculas por separado.

Frase:

Mayúsculas:

Minúsculas:

---

## Ejemplos

### Entrada 0

hoLAmUNDo

### Salida 0

Frase: hoLAmUNDo  
Mayusculas: LAUND  
Minusculas: homo

### Entrada 1

HOLA

### Salida 1

Frase: HOLA  
Mayusculas: HOLA  
Minusculas:

## Solución

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
    char Frase[80];
    char Mayusculas[80];
    char Minusculas[80];

    int ConFrase;
    int ConMayus;
    int ConMinus;

    printf("\nFrase:");
    scanf(" %[^\n]",Frase);

    ConMayus=0;
```

```

ConMinus=0;
ConFrase=0;
while(Frase[ConFrase]!='\0') {

    if (Frase[ConFrase]>='A' && Frase[ConFrase]<='Z') ||
        Frase[ConFrase]=='Ñ') {

        Mayusculas[ConMayus++]=Frase[ConFrase];
        Mayusculas[ConMayus]='\0';
    }

    if ((Frase[ConFrase]>='a' && Frase[ConFrase]<='z') ||
        Frase[ConFrase]=='ñ') {

        Minusculas[ConMinus++]=Frase[ConFrase];
        Minusculas[ConMinus]='\0';
    }

    ConFrase++;
}

printf("\nFrase: %s",Frase);
printf("\nMayusculas: %s",Mayusculas);
printf("\nMinusculas: %s",Minusculas);
printf("\n");
return 0;
}

```

## Capítulo 15

# Verificar Suma

*Tema: condicionales*

Se pide desarrollar un algoritmo que lea tres números y determine si la suma de cualquier pareja de ellos es igual al tercer número. Si se cumple la regla indicada visualizar un mensaje indicando que se cumple y sino indicando que no se cumple.

### Entrada

Se reciben los 3 números por pantalla **Numero1**, **Numero2**, **Numero3**.

### Salida

Mensaje que indica si la comparación de la suma a sido CORRECTA o INCORRECTA.

---

### Ejemplos

#### Entrada 0

1  
2

3

### Salida 0

La comparación ha sido CORRECTA.

### Entrada 1

3

4

5

### Salida 1

La comparación ha sido INCORRECTA.

## Solución

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
    int Numero1=0;
    int Numero2=0;
    int Numero3=0;
    int Existe=0;

    printf("\nTeclee No. 1 : ");
    scanf(" %d",&Numero1);

    printf("Teclee No. 2 : ");
    scanf(" %d",&Numero2);

    printf("Teclee No. 3 : ");
    scanf(" %d",&Numero3);

    if ((Numero1+Numero2)==Numero3 || (Numero1+Numero3)==Numero2 ||
        (Numero2+Numero3)==Numero1) {

        printf("\nLa comparacion ha sido CORRECTA");
```



```
    } else {  
        printf("\nLa comparacion ha sido INCORRECTA");  
    }  
  
    return 0;  
}
```

## Capítulo 16

# Cifrado numérico

*Tema: strings*

Se pide desarrollar un algoritmo que solicite una frase por pantalla y una clave en minúsculas y sin espacios. Una vez aceptada la frase y la clave se ha de visualizar un menú con la siguientes opciones:

- 1) `cifrado`
- 2) `descifrado`
- 3) `salir`

1. Cifrado:

- a) Recibir un mensaje (Texto en Claro, validado solo para minúsculas y sin espacios).  
sms = "david"
- b) Recibir una clave (K).  
K = "hola"  
(En caso que la clave sea más pequeña que el mensaje se rellenará los espacios nuevamente con la clave, hasta que quede empatados cada letra del mensaje con su respectiva letra de la clave).
- c) Se contarán los espacios que hay de la letra del mensaje a la letra de su clave correspondiente (espacios).
  - En el ejemplo se cuenta cuantos espacios hay de la letra "d" para llegar a la letra "h", que es espacios= 4.
  - Si espacios<10, entonces se rellenará con un "0". De manera que cada letra del mensaje sea representado por 2 dígitos. En este caso espacios = "04".
  - Mismo procedimiento para "a" a la "o"
  - Para la letra "v" a la "l" será -> Espacios = 16
  - Para la "i" a la "a" del mensaje será -> Espacios = 18

- Se continúa con el mismo procedimiento hasta finalizar de procesar el mensaje.
  - d) Se mostrara por pantalla el mensaje cifrado: “Mensaje cifrado: 0414161804”
2. Descifrado:
- a) Se recibirá un mensaje cifrado = “0414161804”
  - b) Se recibirá la clave k de descifrado = “hola”.
  - c) Se mostrará por pantalla el texto en claro “david”.

## Entrada

Cifrado:

mensaje

clave

Descifrado:

mensaje

clave

## Salida

Cifrado:

mensaje cifrado

Descifrado:

texto claro

---

## Ejemplos

### Entrada 0

Cifrado:

mensaje: david

clave: hola

## Salida 0

Cifrado:

mensajeCifrado: 0414161804

## Entrada 1

Descifrado:

mensaje: 0414161804

clave: hola

## Salida 1

Descifrado:

textoClaro: david

## Solución

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 27

int menu(){
    printf("\n.....\n");
    printf("1)cifrado\n");
    printf("2)descifrado\n");
    printf("3)salir\n");
    printf(".....\n");
    int op=0;
    scanf("%d",&op);
    return op;
}

void cifrar(char mensaje [MAX], char clave [MAX]){
    //codigo ascii 97=a; 122=z
    int tamanoMensaje = strlen(mensaje);
    int tamanoClave = strlen(clave);
```

```

int indiceMensaje=0;
int indiceClave=0;

int asciiIndiceMensaje;
int asciiIndiceClave;

char mensajeCifrado[MAX*2];
char resultadoStr[3];

mensajeCifrado[0]='\0'; //no es lo mismo "\0" que '\0'
for(indiceMensaje = 0;
    indiceMensaje < tamanoMensaje;
    indiceMensaje++){

    indiceClave = indiceMensaje%tamanoClave;

    asciiIndiceMensaje = mensaje[indiceMensaje];
    asciiIndiceClave = clave[indiceClave];

    int resultado = asciiIndiceClave-asciiIndiceMensaje;
    if(resultado<0){
        resultado = resultado + 26; //26 = tamano abecedario
    }

    itoa(resultado,resultadoStr,10);
    if(resultado<10){
        resultadoStr[0]='\0';
        itoa(resultado,&resultadoStr[1],10);
        resultadoStr[2]='\0';
    }
    strcat(mensajeCifrado,resultadoStr);
}
printf("\nmensajeCifrado: %s\n",mensajeCifrado);
}

void descifrar(char mensajeCifrado [MAX*2], char clave [MAX]){
    //codigo ascii 97=a; 122=z
    int tamanoMensajeCifrado = strlen(mensajeCifrado);
    int tamanoTextoClaro = tamanoMensajeCifrado/2;
    int tamanoClave = strlen(clave);

    int indiceMensajeCifrado=0;
    int indiceClave=0;
    int asciiIndiceClave;

```

```

char textoClaro[MAX];
textoClaro[0]='\0'; //no es lo mismo "\0" que '\0'

char letra[3];
char letraResultante;
int i = 0;
for(indiceMensajeCifrado=0;
    indiceMensajeCifrado<tamanoMensajeCifrado;
    indiceMensajeCifrado+=2
){

    indiceClave = (indiceMensajeCifrado+(tamanoClave-i)) % tamanoClave;
    i++;

    char a = mensajeCifrado[indiceMensajeCifrado];
    char b = mensajeCifrado[indiceMensajeCifrado+1];
    letra[0]=a;
    letra[1]=b;
    letra[2]='\0';

    //convertir char a int
    int letraInt = atoi(letra);
    asciiIndiceClave = clave[indiceClave];
    int resultado = (asciiIndiceClave - letraInt);
    if(resultado<97){
        resultado = resultado + 26;
    }
    letraResultante = resultado;
    strncat(textoClaro,&letraResultante,1);
}
printf("\ntextoClaro: %s\n",textoClaro);
}

int main()
{
    int op=0;
    char mensaje [MAX];
    char clave [MAX];
    while(op!=3){
        op=menu();
        switch(op){
            case 1:
                printf("mensaje: ");
                scanf("%s",mensaje);
                printf("clave: ");

```

```

        scanf("%s",clave);
        cifrar(mensaje,clave);
        break;
    case 2:
        printf("mensaje: ");
        scanf("%s",mensaje);
        printf("clave: ");
        scanf("%s",clave);
        descifrar(mensaje,clave);
        break;
    case 3:
        printf("\nfin del programa\n");
        break;
    default:
        printf("\nopcion incorrecta\n");
        break;
    }
}
return 0;
}

```

## Capítulo 17

# Búsqueda en anchura: alcance más corto

*Temas: grafos*

Dado un grafo no dirigido que consta de  $N$  nodos (etiquetados 1 a  $N$ ) , donde un determinado nodo  $S$  dado representa la posición de inicio y un borde entre dos nodos es de longitud 6 unidades en el grafo.

Se requiere para el cálculo de la distancia más corta desde la posición de inicio ( nodo  $S$  ) para todos los otros nodos en el grafo.

- Si un nodo es inalcanzable, la distancia se asume como -1.
- La longitud de cada borde en el grafo es de 6 unidades.

## Restriciones

$$1 \leq T \leq 10$$

$$2 \leq N \leq 1000$$

$$1 \leq M \leq (N \times (N-1))/2$$

$$1 \leq x, y, S \leq N$$

## Entrada

En la primera línea se ingresa un entero  $T$ , el cual va ser el número de casos de prueba que se va a realizar.



La primera línea de cada caso de prueba tiene dos enteros  $N$  , que denota el número de nodos en el grafo y  $M$  , que indica el número de aristas en el grafo.

Las siguientes líneas  $M$  se componen cada uno de dos números enteros separados por espacios  $x$  y, donde  $x$  e  $y$  denotan los dos nodos entre los que existe el borde .

la última línea de un caso de prueba tiene un entero  $S$  , que indica la posición de partida .

## Salida

Para cada uno de los casos de prueba  $T$ , imprimir una sola línea que consiste en  $N - 1$  enteros separados por espacios , que denota las distancias más cortas de los nodos  $N - 1$  desde la posición de partida  $S$ . Esto se hará para todos los mismos nodos como en el orden de entrada 1 a  $N$ .

Para los nodos inalcanzables, se imprime -1.

---

## Ejemplos

### Entrada 0

```
2
4 2
1 2
1 3
1
3 1
2 3
2
```

### Salida 0

```
6 6 -1
```

## Solución

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define DESDE 0
#define TO 1
#define DISTANCIA 0
#define ESTADO 1
#define AGOTADO 0
#define PENDIENTE 1
#define ESP 6

int main() {
    int casos, nodos, bordes, activado, i, origen;
    int nodo[1000+1][2];
    int borde[1000*(1000-1)/2+1][2];

    scanf("%d", &casos);

    while(casos--){
        scanf("%d%d", &nodos, &bordes);
        for(i=1; i <= bordes; i++){
            scanf("%d%d", &borde[i][DESDE], &borde[i][TO]);
        }
        for(i=1; i <= nodos; i++){
            nodo[i][DISTANCIA] = -1;
            nodo[i][ESTADO] = PENDIENTE;
        }
        scanf("%d", &origen);
        activado = origen;
        nodo[origen][DISTANCIA] = 0;
        unsigned int min;
        while(activado){
            activado = 0;
            min = -1;
            for(i=1; i <= nodos; i++){
                if (nodo[i][ESTADO]==PENDIENTE &&
```

```

        nodo[i][DISTANCIA] != -1 &&
        nodo[i][DISTANCIA] < min
    ){

        min = nodo[i][DISTANCIA];
        activado = i;
    }
}

for(i=1; i<=bordes; i++){
    if (borde[i][DESDE] == activado){
        if (nodo[borde[i][TO]][DISTANCIA] >
            nodo[borde[i][DESDE]][DISTANCIA]+ESP ||
            nodo[borde[i][TO]][DISTANCIA]==-1
        ){
            nodo[borde[i][TO]][DISTANCIA] =
            nodo[borde[i][DESDE]][DISTANCIA]+ESP;
        }
    }

    if (borde[i][TO] == activado){
        if(nodo[borde[i][DESDE]][DISTANCIA] >
            nodo[borde[i][TO]][DISTANCIA]+ESP ||
            nodo[borde[i][DESDE]][DISTANCIA]==-1
        ){
            nodo[borde[i][DESDE]][DISTANCIA] =
            nodo[borde[i][TO]][DISTANCIA]+ESP;
        }
    }
}

nodo[activado][ESTADO] = AGOTADO;
}

for(i=1; i<=nodos; i++){
    if(i!=origen)printf("%i ", nodo[i][DISTANCIA]);
}
printf("\n");
}
return 0;
}

```

## Capítulo 18

# Palabra rangoli

*Temas: bucles, arreglos*

**Rangoli** es una forma de arte popular de la India basada en la creación de patrones. Se desea realizar un algoritmo (Desarrollado en Lenguaje C) el cual muestre en pantalla como el siguiente ejemplo.

```
Ingresa la palabra:
>> hola
-----a-----
----a-l-a----
--a-l-o-l-a--
a-l-o-h-o-l-a
--a-l-o-l-a--
----a-l-a----
-----a-----
```

## Entrada

El dato de entrada que se debe ingresar es una palabra (arreglo de caracteres) con el cual se desea formar la figura.

## Salida

Se debe de mostrar por pantalla la figura antes mencionada construida a base de la palabra que se pidió ingresar por teclado.

---

## Ejemplos

### Entrada 0

Juan Perez

### Salida 0

```
Ingrese la palabra:
>> Juan Perez

-----Z-----
-----Z-e-Z-----
-----Z-e-r-e-Z-----
-----Z-e-r-e-r-e-Z-----
-----Z-e-r-e-P-e-r-e-Z-----
-----Z-e-r-e-P- -P-e-r-e-Z-----
-----Z-e-r-e-P- -n- -P-e-r-e-Z-----
----Z-e-r-e-P- -n-a-n- -P-e-r-e-Z----
--Z-e-r-e-P- -n-a-u-a-n- -P-e-r-e-Z--
Z-e-r-e-P- -n-a-u-J-u-a-n- -P-e-r-e-Z
--Z-e-r-e-P- -n-a-u-a-n- -P-e-r-e-Z--
----Z-e-r-e-P- -n-a-n- -P-e-r-e-Z----
-----Z-e-r-e-P- -n- -P-e-r-e-Z-----
-----Z-e-r-e-P- -P-e-r-e-Z-----
-----Z-e-r-e-P-e-r-e-Z-----
-----Z-e-r-e-r-e-Z-----
-----Z-e-r-e-Z-----
-----Z-e-Z-----
-----Z-----
```

---

## Solución

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void vaciar_buffer() {
    int ch;
    while ((ch = getchar()) != '\n' && ch != EOF);
}

void construir_rangoli(char *palabra) {
    int i, j, k;
    int guion = (strlen(palabra)*2)-2;
    int factor, v = 0;

    if (strlen(palabra) == 1)
        printf("\n %s", palabra);
    else {
        for (i=0; i<(strlen(palabra)*2)-1; i++) {
            k = 1;
            for (j=0; j<(((strlen(palabra)*2)-1)*2)-1; j++) {
                if (j != guion) {
                    putchar('-');
                } else {
                    putchar(palabra[strlen(palabra)-k]);
                    if (j < (strlen(palabra)*2)-2) {
                        guion+=2;
                        k++;
                    } else if (j == (strlen(palabra)*2)-2) {
                        factor = guion - 2*k;
                        k--;
                        if (k != 0)
                            guion+=2;
                    } else {
                        k--;
                        if (k != 0)
                            guion+=2;
                    }
                }
            }
        }
        if (factor >= 0 && v == 0)
```

```

        guion = factor;
    else if (v == 0) {
        guion = 2;
        v++;
    } else
        guion = factor + 4;
    putchar('\n');
}
}

int main() {
    char *palabra = (char *)malloc(15 * sizeof(char));

    printf("\n Ingrese la palabra:\n >> ");
    fgets(palabra, 16 * sizeof(char), stdin);
    if (palabra[strlen(palabra)-1] == '\n' )
        palabra[strlen(palabra)-1] = '\0';
    else
        vaciar_buffer();

    construir_rangoli(palabra);
    return 0;
}

```

## Capítulo 19

# Sherlock Holmes y la Bestia

***Temas:** bucles, arreglos*

Sherlock Holmes sospecha que su archienemigo, el profesor Moriarty, es una vez más está tramando algo diabólico. El compañero de Sherlock, el Dr. Watson, sugiere Moriarty puede ser el responsable de los últimos números del MI6 con su super-computador, la bestia.

Poco después, Sherlock recibe una nota de Moriarty alarde de infectar La bestia con un virus; Sin embargo, también le da una pista: un número,  $N$ . Sherlock determina la clave para eliminar el virus es encontrar el mayor **número decente** que tiene  $n$  dígitos.

Un **número decente** tiene las siguientes propiedades:

1. Los dígitos solo pueden ser 3 y/o 5.
2. El número que contiene 3, solo es divisible para 5.
3. El número que contiene 5, solo es divisible para 3.
4. Si hay más de uno de esos números, se elige el más grande.

El **virus** de Moriarty tiene un reloj de cuenta regresiva para la destrucción de la bestia, y el tiempo se acaba rápido. La tarea es ayudar a Sherlock encontrar la **llave** antes de la bestia es destruida!.

## Entrada

En la primera línea se ingresa un entero  $T$ , el cual va ser el número de casos de prueba que se va a realizar.

Las líneas posteriores  $T$  contienen cada uno un número entero,  $N$ , detallando el número de dígitos del número.



## Salida

Imprimir el mayor número decente que tiene N dígitos; si no existe tal número, informe a Sherlock mediante impresión -1.

---

## Ejemplos

### Entrada 0

4  
1  
3  
5  
11

### Salida 0

-1  
555  
33333  
55555533333

---

## Solución

```
#include <stdio.h>
#include <string.h>

void largestDecentNumber(int digits){
    int found = -1;
    for(int i = 0; i <= digits; ++i){
        int aux = digits-i;
        if((aux%5 + i%3) == 0) found = i;
    }
    if(found == -1) printf("-1");
}
```

```

    else{
        for(int i = 0; i < found; ++i) printf("5");
        for(int i = found; i < digits; ++i)printf("3");
    }
    return;
}
int main() {
    int T;
    scanf("%d",&T);
    for(int i = 0; i < T; ++i){
        int N;
        scanf("%d",&N);
        largestDecentNumber(N);
        printf("\n");
    }
    return 0;
}

```

## Capítulo 20

# Mensajes cifrados

*Temas: bucles, string, arrays*

Batman está bajo la pista de Enigma, quien ha dejado mensajes cifrados por toda ciudad Gótica. Pronto Batman descubre que los mensajes pueden ser descifrados de la siguiente manera:

Se tiene un texto **T**, donde **L** es la longitud de **T**. Se ingresa cada carácter en un array bidimensional, tal que sus filas y columnas cumplan con lo siguiente:

El mensaje descifrado es obtenido mostrando los caracteres de cada columna del array, seguido por un espacio, y así hasta la última columna. Ayuda a Batman a averiguar qué es lo que trama Enigma.

### Entrada

La primer línea contiene la cadena de texto cifrada, la cual consta de sólo letras del abecedario inglés sin espacios. La longitud del texto es menor a 90.

### Salida

Mostrar el mensaje descifrado.

Nota: Para las funciones entero mayor y entero menor se pueden usar las funciones `ceil()` y `floor()` disponibles en la librería `<math.h>`

## Ejemplos

### Entrada 0

dtneiatpcpmooiaedlar

### Salida 0

depar tame ntod epol icia

### Explicación

La longitud del mensaje es de 21 caracteres, por lo tanto el número el número de filas y columnas del array es de 5 y 5 respectivamente.

### Entrada 1

htnddtoateaislrdecpcaigaieuluo

### Salida 1

hospi talce ntral deciu dadgo tica

---

## Solución

```
#include <stdio.h>
#include <math.h>
#include <string.h>

int main() {
    char mensaje_cifrado[90];
    int long_texto;
    int num_columnas;
    int i;

    printf("Mensaje cifrado: "); scanf("%s", mensaje_cifrado);

    long_texto = strlen(mensaje_cifrado);
```

```

num_columnas = ceil(pow(long_texto, 0.5));

printf("Texto descifrado:\n");
for (int columna = 0; columna < num_columnas; columna++) {
    i = columna;
    while (i < long_texto) {
        printf("%c", mensaje_cifrado[i]);
        i += num_columnas;
    }
    printf(" ");
}
printf("\n");

return 0;
}

```

## Capítulo 21

# Criba de Erastótenes

***Temas:** arrays, matemáticas, números primos*

La criba de Eratóstenes es un algoritmo que permite hallar todos los números primos menores que un número natural  $N$ . Se forma de una tabla con todos los valores comprendidos entre  $2$  y  $N$ , y se van tachando los números que no son primos de la siguiente manera: Comenzando por el  $2$ , se tachan todos sus múltiplos. Se escoge el siguiente número que no esté tachado, y se tachan sus múltiplos. Así sucesivamente hasta que el cuadrado del número sea mayor que  $N$ .

### Entrada

La primer línea contiene un entero menor a  $500\ N$ , que es el número límite.

### Salida

Una lista de todos los números primos encontrados separados por un espacio.

---

## Ejemplos

### Entrada 0

25

### Salida 0

2 3 5 7 11 13 17 19 23

### Explicación

Los números primos entre 1 y 25 son:

2, 3, 5, 7, 11, 13, 17, 19, 23.

### Entrada 1

113

### Salida 1

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89  
97 101 103 107 109 113.

---

## Solución

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

const int MAX = 500;

void inicializar_array(bool primos[]);
void encontrar_primos(bool primos[], int numero_limite);
void tachar_multiplos(bool primos[], int numero_limite, int num);
void mostrar_primos(bool primos[], int numero_limite);

int main() {
```

```

    bool primos[MAX];
    int numero_limite;
    printf("Ingrese el numero limite: "); scanf("%d", &numero_limite);
    inicializar_array(primos);
    encontrar_primos(primos, numero_limite);
    mostrar_primos(primos, numero_limite);
    return 0;
}

void inicializar_array(bool primos[]){
    for (int i = 0; i < MAX; i++){
        primos[i] = true;
    }
}

void encontrar_primos(bool primos[], int numero_limite){
    for(int i = 2; i*i <= numero_limite; i++){
        if (primos[i]){
            tachar_multiplos(primos, numero_limite, i);
        }
    }
}

void tachar_multiplos(bool primos[], int numero_limite, int num){
    for(int i = num*2; i <= numero_limite; i += num){
        primos[i] = false;
    }
}

void mostrar_primos(bool primos[], int numero_limite){
    for (int i = 2; i <= numero_limite; i++){
        if(primos[i]){
            printf("%d ", i);
        }
    }
    printf("\n");
}

```



## Capítulo 22

# Cifrado de vigenere

***Temas:** aritmetica modular, strings, arrays.*

El cifrado de vigenere está basado en el cifrado César, a diferencia de éste que sólo usa un número como clave, el cifrado de vigenere usa una palabra como clave. Su funcionamiento es el siguiente:

Se numera el alfabeto del 1 al 26. A cada letra se asigna su correspondiente número. Se tiene un mensaje **S** y una palabra clave **K**, se sitúa la clave debajo del mensaje a cifrar, si el mensaje es más largo que la clave, esta se repite hasta el fin del texto. Se desplazan los caracteres de **S** por el valor numérico correspondiente al carácter de **K** que está debajo del carácter de **S**.

### Entrada

La primer línea contiene un string **S**, que es la cadena a cifrar y cuya longitud no pasa de los 100 caracteres y sólo contiene letras minúsculas del abecedario inglés. La segunda línea es un string que representa la clave **K**, cuya longitud no pasa de los 50 caracteres (sólo letras minúsculas del abecedario inglés).

### Salida

Mostrar la cadena cifrada.

---

## Ejemplos

### Entrada 0

```
holamundo
uno
```

### Salida 0

```
ccavajird
```

## Explicación

Si colocamos la clave “uno” debajo del mensaje a cifrar, tenemos:

Si reemplazamos cada carácter de la clave por su valor numérico, se tiene:

Si desplazamos cada carácter del texto a cifrar el número dado en la tabla anterior, es decir, a “h” la desplazamos 22 posiciones (volviendo a empezar desde la “a” en caso de sobrepasar la “z”), dando “c” como resultado. Realizamos este procedimiento con el resto de letras, al final se tiene que el mensaje cifrado es “ccavajird”.

### Entrada 1

```
ghdjakzzzxsdpweowsdasfdsaasdfp  
lilixxz
```

### Salida 1

```
sqpsyizlijbbnwqxibbysrmejyqdrub  
bnu
```

### Entrada 2

```
qwertyuiopasdfghjklmnbcxz  
qqaazz
```

## Salida 2

hnfstylzpqasuwhijkdocvcoq

---

## Solución

```
#include <stdio.h>
#include <string.h>

const int MAX_MSJ = 101;
const int MAX_CLAVE = 51;

int main() {
    char mensaje[MAX_MSJ];
    char clave[MAX_CLAVE];
    char mensaje_cifrado[MAX_MSJ];
    int longitud_cadena;
    int longitud_clave;

    printf("Ingrese la cadena: "); gets(mensaje);
    printf("Ingrese la clave: "); gets(clave);

    longitud_cadena = strlen(mensaje);
    longitud_clave = strlen(clave);

    int j = 0;
    int i = 0;
    for(i = 0; i < longitud_cadena; i++) {
        mensaje_cifrado[i] = (mensaje[i] + clave[j] - 193)%26 + 97;
        j++;
        if(j >= longitud_clave) {
            j = 0;
        }
    }
    mensaje_cifrado[i] = '\0';
    printf("Mensaje cifrado: %s\n", mensaje_cifrado);
    return 0;
}
```

## Capítulo 23

# Cuys

***Temas:** matrices, recursividad, programación dinámica*

De la granja de Juan Pérez se escaparon  $M$  cuys, se tiene una carretera con  $N$  carriles y una distancia de la vía  $D$ , se ingresa las posiciones de los  $M$  cuys con referencia a la vía, se conoce que dicha vía se divide por bloques y en cada bloque puede estar o no un cuy. El reto es encontrar el número de caminos que existen para llegar al extremo de la vía sin matar a ningún cuy. Sabiendo que los movimientos posibles del conductor son 3: hacia arriba, derecha; hacia abajo, derecha; y hacia la derecha.

## Entrada

La primer línea contiene el número de cuys, la segunda línea el número de vías, y la tercer línea la longitud de la carretera. Luego se tiene  $M$  líneas que indican la posición de un cuy dentro de la carretera.

## Salida

El número de caminos existentes para llegar a la otra vía sin matar a un cuy, si se parte desde la esquina superior de la carretera (posición 0, 0).

---

## Ejemplos

### Entrada 0

```
3
4
5
0, 3
1, 1
2, 4
```

### Salida 0

```
6
```

### Explicación

Existen 6 caminos diferentes que se pueden tomar sin matar a ningún  
cuy partiendo desde la posició (0, 0).

---

## Solución 1

Solución simple, usando recursividad.

```
#include <stdio.h>
#include <stdlib.h>

int num_filas = 4, num_columnas = 5;

int matriz[5][5] = {{0, 0, 0, 1, 0},
                    {0, 1, 0, 0, 0},
                    {0, 0, 0, 0, 1},
                    {0, 0, 0, 0, 0}
};

int caminos(int fila, int columna) {
    int contador = 0;
```

```

    if (columna == num_columnas - 1) {
        return 1;
    }

    if (fila != 0 && matriz[fila - 1][columna + 1] != 1) {
        contador += caminos(fila - 1, columna + 1);
    }

    if (matriz[fila][columna + 1] != 1) {
        contador += caminos(fila, columna + 1);
    }

    if (fila != num_filas - 1 && matriz[fila + 1][columna + 1] != 1) {
        contador += caminos(fila + 1, columna + 1);
    }

    return contador;
}

int main(int argc, char const *argv[]) {
    int numero_caminos;
    numero_caminos = caminos(0, 0);
    printf("%d\n", numero_caminos);

    return 0;
}

```

## Solución 2

Solución mejorada, usando programación dinámica.

```

#include <stdio.h>
#include <stdlib.h>

int num_filas = 4, num_columnas = 5;

int matriz[5][5] = {{0, 0, 0, 1, 0},
                    {0, 1, 0, 0, 0},
                    {0, 0, 0, 0, 1},
                    {0, 0, 0, 0, 0}
};

```

```

int cache[5][5] = {{-1, -1, -1, -1, -1},
                  {-1, -1, -1, -1, -1},
                  {-1, -1, -1, -1, -1},
                  {-1, -1, -1, -1, -1}
                  };

int caminos(int fila, int columna) {
    int contador = 0;

    if (cache[fila][columna] != -1) {
        return cache[fila][columna];
    }

    if (columna == num_columnas - 1) {
        return 1;
    }

    if (fila != 0 && matriz[fila - 1][columna + 1] != 1) {
        contador += caminos(fila - 1, columna + 1);
    }

    if (matriz[fila][columna + 1] != 1) {
        contador += caminos(fila, columna + 1);
    }

    if (fila != num_filas - 1 && matriz[fila + 1][columna + 1] != 1) {
        contador += caminos(fila + 1, columna + 1);
    }

    cache[fila][columna] = contador;
    return contador;
}

int main(int argc, char const *argv[]) {
    int numero_caminos;
    numero_caminos = caminos(0, 0);
    printf("%d\n", numero_caminos);

    return 0;
}

```