

Matrices I**Ejercicio 1. Escribe, compile y ejecute los siguientes programas.****Programa 1:**

El programa 1 es un acercamiento inicial a las matrices en el lenguaje C. Lo que hace este programa es imprimir el valor de cada una de las celdas de la matriz que hemos ingresado.

Primeramente, abrimos el *main* y declaramos la variable entera *i* y la definimos como “0”, y también declaramos la matriz de un renglón y ocho columnas (*columna[8]*). Posteriormente, definimos el valor de cada una de las ocho columnas, siendo la primera “*columna[0]=0*”, la segunda “*columna[1]=1*”, la tercera “*columna[2]=2*” y así sucesivamente hasta llegar a la octava columna definida como “*columna[7]=7*”. Después, simplemente ocupamos el ciclo *for* para imprimir el valor de cada una de las columnas mediante este formato: “*El valor de la celda %u es %u*” donde *%u* se utiliza para imprimir un *entero sin signo*, y en donde la primera *%u* imprime el número de la columna y el segundo *%u* imprime el valor definido a esa columna.

El ciclo *for* se ha especificado mediante la condición que la variable entera *i* debe ser menor a 8, en un principio esta variable esta definida como “*i=0*” pero al empezar el ciclo *for* la variable queda definida como “*i=1*” y esta incrementará su valor en uno (es decir, se le sumará 1) cada vez que el ciclo vuelva a repetirse y se detendrá cuando ya no se cumpla la condición del ciclo (es decir, cuando *i* ya no sea menor que 8).

Finalmente cerramos el ciclo *for* y cerramos el programa con *return 0*. Así, el código queda de la siguiente manera:

```
//El siguiente programa muestra el llenado de una matriz mediante la asignación
//directa. Todas las variables que componen la matriz tienen el mismo nombre "semana"
//pro diferente índice (el valor del argumento i=0.1.2...7)

#include <stdio.h>

int main(void){

    int semana[8];
    int i=0;

    semana[0]=0;
    semana[1]=1;
    semana[2]=2;
    semana[3]=3;
    semana[4]=4;
    semana[5]=5;
    semana[6]=6;
    semana[7]=7;

    for(i=1;i<8;i++){
        printf("El valor de la celda %u es %u\n",i,semana[i]);
    }

    return 0;
}
```

Compilamos y ejecutamos el programa y lo que hace es imprimir el número de la columna de la matriz y el valor asociada a ella. En este caso, todo queda perfecto para que se imprima el valor de todas las columnas.

```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz.c  Plantillas  Vídeos
Desktop    examples.desktop  Matriz1.c  Música    Público
ubuntu@ubuntu:~$ gcc -o matriz matriz.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Matriz1.c  Música  Público
Desktop    examples.desktop  matriz  matriz.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ ./matriz
El valor de la celda 1 es 1
El valor de la celda 2 es 2
El valor de la celda 3 es 3
El valor de la celda 4 es 4
El valor de la celda 5 es 5
El valor de la celda 6 es 6
El valor de la celda 7 es 7
ubuntu@ubuntu:~$

```

Programa 2:

El código de este programa se escribe de manera similar al anterior, únicamente que ahora declaramos la matriz de un renglón y ocho columnas como *char* y definimos cada columna de la matriz como un carácter. En este caso, definimos el valor de cada una de las ocho columnas como “*columna[0]='a'*” a la primera columna, la segunda como “*columna[1]='b'*”, la tercera “*columna[2]='c'*” y así sucesivamente hasta llegar a la octava columna definida como “*columna[7]='h'*”. También se modificó la función *printf* para imprimir el valor de cada columna, ya que estos valores al ser de tipo *char* se necesitan especificar con *%c* (que imprime caracteres).

Un hecho a destacar es que para asignar un número a una variable se colocan los números directamente (ejemplo, *int i=12*), sin embargo, al momento de asignar caracteres de hace mediante el uso de las comillas simples (ejemplo, *char arg='a'*).

El código quedó de la siguiente manera:

```

//Este programa hacerlo mismo pero ahora con caracteres.

#include <stdio.h>

int main(void){

    char semana[8];
    int i=0;

    semana[0]='a';
    semana[1]='b';
    semana[2]='c';
    semana[3]='d';
    semana[4]='e';
    semana[5]='f';
    semana[6]='g';
    semana[7]='h';

    for(i=1;i<8;i++){
        printf("El valor de la celda %u es %c\n",i,semana[i]);
    }

    return 0;
}

```

Compilamos y ejecutamos el programa y lo que hace es imprimir el número de la columna de la matriz y el valor asociada a ella, que en este caso es un carácter.

```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz1.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz1.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz1.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz  Música  Público
ubuntu@ubuntu:~$ ./matriz
El valor de la celda 1 es b
El valor de la celda 2 es c
El valor de la celda 3 es d
El valor de la celda 4 es e
El valor de la celda 5 es f
El valor de la celda 6 es g
El valor de la celda 7 es h
ubuntu@ubuntu:~$

```

Programa 3:

Este es un programa sencillo donde se utilizan las matrices para sacar la distancia de un objeto en sus ejes coordenados (es decir, su distancia en x y en y) a partir de las velocidades que tiene en los ejes coordenados y el tiempo que recorrió durante un trayecto determinado.

Primeramente, abrimos el *main* y declaramos las variables. Primero declaramos la matriz de un renglón y dos columnas “*velocidad[2]*” y definimos sus columnas como “*velocidad[2]={4.55, 6.73}*” es decir, que la columna *velocidad[0]=4,55* y la columna *velocidad[1]=6,73*. Después declaramos las variables flotantes *distanciay* y *distanciay* y ambas las definimos como 0. También declaramos la variable flotante *tiempo* y la definimos como “*tiempo=3.00*” y declaramos la matriz de un renglón y dos columnas “*coordenada[2]={'x', 'y'}*”, es decir, la columna 1 de esta matriz queda definida como *coordenada[0]='x'* y la columna 2 queda definida como *coordenada[1]='y'*.

Después definimos que la variable “*distanciay*” es igual al producto “*velocidad[0]*tiempo*”, es decir, la primer columna de la matriz *velocidad[2]* multiplicada por la variable *tiempo*. De igual manera, la variable “*distanciay*” se definió como el producto entre la segunda columna dela matriz *velocidad[2]* y la variable *tiempo*.

Por ultimo, se utilizo la función *printf* para imprimir la coordenada y la distancia en esa coordenada. El código quedo dela siguiente manera:

```

//llenado de una matriz con dato, una sola vez

#include <stdio.h>

int main(void){

    float velocidad[2]={4.55,6.73};
    float distanciay=0.00;
    float distanciay=0.00;
    float tiempo=3.00; //media horas
    char coordenada[2]='x','y';

    distanciay=velocidad[0]*tiempo;
    distanciay=velocidad[1]*tiempo;

    printf("La distancia en %c es %f\n",coordenada[0], distanciay);
    printf("\nLa distancia en %c es %f\n",coordenada[1], distanciay);

    return(0);
}

```

Compilamos y ejecutamos el programa y lo que hace es imprimir las frases “La distancia en x es %f” donde x es eje coordenado x y %f es la distancia en la coordenada x. Y la frase “La distancia en y es %f” donde y es eje coordenado y y %f es la distancia en la coordenada y.

```
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz2.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz2.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz2.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz  Música  Público
ubuntu@ubuntu:~$ ./matriz
La distancia en x es 13.650001
La distancia en y es 20.190001
ubuntu@ubuntu:~$
```

Programa 4:

Este programa sirve para multiplicar un vector por un escalar.

Primero abrimos el *main* y declaramos las variables. En esta ocasión declaramos la matriz de un renglón y dos columnas “*vector[2]={0,0}*” la cual será nuestro vector, también declaramos la matriz de un renglón y dos columnas “*campo[2]={0,0}*” que usaremos para guardar las variables que salen como resultado del producto del vector con un escalar. Y declaramos la variable flotante “*escalar=0,000*” que será nuestra variable que usaremos como escalar.

Una vez declaradas y definidas, utilizamos la función *puts* para imprimir una cadena de caracteres (muy similar a la función *printf*) que pida introducir el valor de la componente x del vector y con la función *scanf* guardamos el numero introducido en la variable *vector[0]* (La cual es la primera columna de nuestra matriz *vector[2]*) y hacemos lo mismo con la componente y del vector (usamos *puts* para imprimir una frase que pida el valor de la componente y del vector y luego usamos *scanf* para guardar ese numero en la variable *vector[1]* que es la segunda columna de nuestra matriz *vector[2]*).

Después utilizamos la función *printf* para observar el vector que se ha introducido y nuevamente usamos *puts* para pedir, ahora, el factor de estiramiento o el numero escalar con el que se va a efectuar la multiplicación por escalar y nuevamente usamos *scanf* para guardar el numero introducido en la variable *escalar*.

Ahora, efectuamos las operaciones de la siguiente manera:

Definimos la primera columna de *campo[2]* como “*campo[0]=vector[0]*escalar*” y, similarmente, definimos la segunda columna de *campo[2]* como “*campo[1]=vector[1]*escalar*”.

Por ultimo, usando la función *printf* imprimimos la frase “El vector estirado en %g es [%g,%g]”, donde el primer %g es la variable *escalar* y los otros dos %g son *campo[0]* y *campo[1]* respectivamente.

Por ultimo, usamos *puts* para imprimir en pantalla “Oprime return para salir” y usamos la función *getchar()* para capturar ese enter y pasar a *return 0* donde termina el código.

El código de este programa quedo de la siguiente manera:

```

//Obtendremos un vector estirado
#include <stdio.h>

int main(void){

    float vector[2]={0,0};
    float escalar=0.000;
    float campo[2]={0.000,0.000};

    puts("Escribe el componente x del vector: ");
    scanf("%f", &vector[0]);
    puts("Escribe el componente y del vector: ");
    scanf("%f", &vector[1]);

    printf("El vector introducido es [%g,%g]\n\n",vector[0],vector[1]);

    puts("Escribe, ahora, el factor de estiramiento: ");
    scanf("%f", &escalar);

    campo[0]=escalar*vector[0];
    campo[1]=escalar*vector[1];

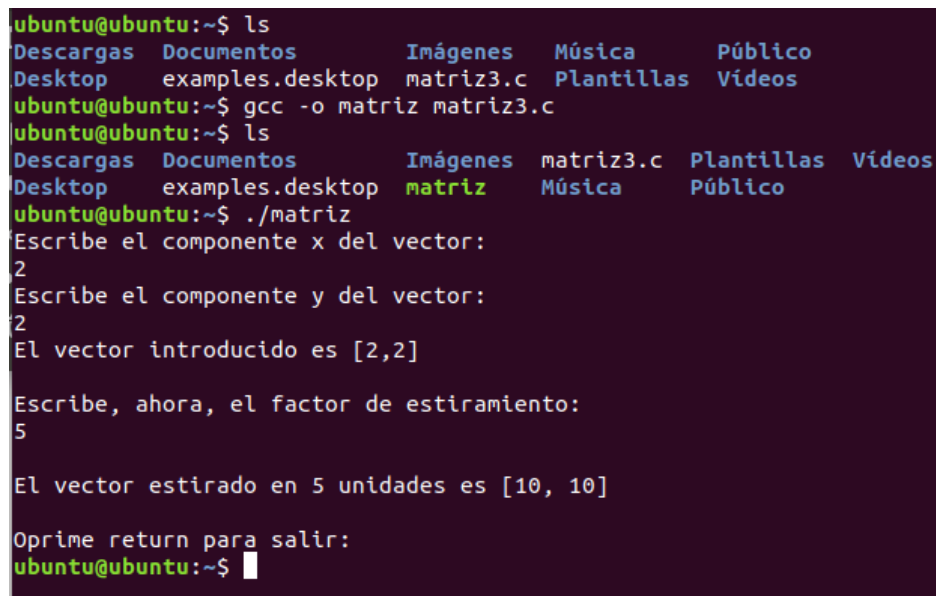
    printf("\nEl vector estirado en %g unidades es [%g, %g]\n\n", escalar, campo[0], campo[1]);

    puts("Oprime return para salir: ");
    getchar();

    return(0);
}

```

Compilamos y ejecutamos el programa y lo que hace es pedir las componentes x y y de un vector, lo imprime en pantalla y luego pide el factor de estiramiento (para hacer el producto por escalar) y después imprime “El vector estirado en %g es [%g,%g]”, donde el primer %g es la variable *escalar* y los otros dos %g son *campo[0]* y *campo[1]* respectivamente que son las componentes del vector estirado. Luego se imprime “Presione return para salir” y se espera a que introduzcamos *return* para terminar el programa.



```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz3.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz3.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz3.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz    Música    Público
ubuntu@ubuntu:~$ ./matriz
Escribe el componente x del vector:
2
Escribe el componente y del vector:
2
El vector introducido es [2,2]

Escribe, ahora, el factor de estiramiento:
5

El vector estirado en 5 unidades es [10, 10]

Oprime return para salir:
ubuntu@ubuntu:~$

```

Programa 5.

Este ejercicio es cortito, consiste en un programa que imprime todos los valores de todas las columnas de una matriz de un renglón y de n columnas, donde n es natural, en nuestro caso sera para una matriz de 7 columnas.

Primero abrimos el *main* y declaramos la variable entera “*j=0*” y la matriz “*semana[7]={'d','l','m','i','j','v','s'}*”. Después creamos un ciclo *for* donde defina a *j* como 1 y que se ejecute cuando la variable *j* sea menor que 7 y que en cada vuelta *j* incremente uno. Este ciclo va a imprimir el numero de la columna correspondiente junto con su valor.

Terminamos el código y este quedo así:

```
//Programa paramostrar el llenado portecclado de una matriz

#include <stdio.h>

int main(void){

    char semana[7]={'d','l','m','i','j','v','s'}; //matriz a llenar
    int j=0; //contador

    for(j=0;j<7;j++){
        printf("Celda no. %u=%c\n",j,semana[j]);
    }

    return(0);
}
```

Compilamos y ejecutamos el programa y lo que hace es imprimir el numero de columna de la matriz junto con su valor correspondiente.



```
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz4.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz4.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz4.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz  Música  Público
ubuntu@ubuntu:~$ ./matriz
Celda no. 0=d
Celda no. 1=l
Celda no. 2=m
Celda no. 3=i
Celda no. 4=j
Celda no. 5=v
Celda no. 6=s
ubuntu@ubuntu:~$
```

Programa 6.

Este ejercicio es aún más cortito, consiste en un programa que imprime todos los valores de todas las columnas de una matriz de un renglón y de n columnas, donde n es natural pero en forma de cadena por lo que la cadena debe determinar en carácter nulo, es decir, en cero. En nuestro caso sera para una matriz de 8 columnas.

Primero abrimos el *main* y declaramos la variable entera "*i=0*" y la matriz "*semana[8]={'d','l','m','i','j','v','s','0'}*", nótese que esta matriz termina en carácter nulo. Después, mediante la función *printf* hacemos que se imprima en pantalla la palabra "*Cadena*" seguida de la cadena del valor de todas las columnas de la matriz (en nuestro caso: *dlmijvs*).

El código del programa quedo dela siguiente manera:


```

//Esta es otra forma de inicializar una matriz de caracteres
//pero para que forme una cadena.
//Ahora la cadena debe de terminar con el caracter nulo.

#include <stdio.h>

int main(void){

    char semana[8]={ 'd','l','m','i','j','v','s',0}; //matriz a llenar
    int i=0; //contador

    printf("Cadena %s\n",semana);
    |
    return(0);
}

```

Compilamos y ejecutamos el programa y lo que hace es imprimir la palabra “Cadena” seguida de todos los valores de todas las columnas (por orden) de la matriz.

```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz5.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz5.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz5.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz  Música  Público
ubuntu@ubuntu:~$ ./matriz
Cadena dlmijvs
ubuntu@ubuntu:~$ █

```

Programa 7.

Para este programa primero abrimos el *main* y declaramos las variables enteras *i,j,k* y la matriz “*semana[7]={ 'l','m','i','j','v','s','d' }*” además del puntero “*dia*” de tipo *char*. Después iniciamos un ciclo *for* definiendo “*j=0*” y bajo la condición de que *j* sea menor que 7 y que cada vez que se repita el ciclo, se le sumará 1 a *j*. Este ciclo va a imprimir “*El dia es:* ” seguido de la letra correspondiente a la matriz. Como es un ciclo se repetirá hasta que “*j=8*” cuando se detendrá y *getchar()* esperará a que o introduzcamos un carácter o que presionemos enter. Después de ello igualamos el puntero *dia* con la matriz *semana[7]* y lo imprimimos en pantalla mediante *printf*. Después hay un *getchar()* que nuevamente esperara a que introduzcamos un carácter o presionemos enter y después hay otro ciclo *for*, Este ciclo define “*k=1*” y se ejecutara siempre que *k* sea menor a 8 y cada vez que el ciclo se repita se le sumara 1 a la variable *k*. Este ciclo imprimirá “*La dirección %u es %u*” donde la primer %u es el correspondiente a *k+1* y la segunda %u a *dia+1*.

Al final el código queda de la siguiente manera:

```

#include <stdio.h>

int main(void){

    char semana[7]={'l','m','i','j','v','s','d'};
    int i, j, k;
    char dia; //declaramos el puntero dia

    for(j=0;j<7;j++){
        printf("El dia es %c\n", semana[j]);
    }

    //Ahora se va a guardar en el puntero dia la direccion de la matriz semana
    dia=semana[7];

    //Note que en la direccion anterior no colocamos el operador de indicación "&"
    //Vamos a imprimir el contenido de dia
    printf("El puntero dia contiene la direccion: %u\n", dia);
    getchar();

    //Ahora vamos a imprimir el resto de direcciones correspondientes a
    //los elementos siguientes.
    for(k=1;k<8;k++){
        printf("La direccion %u es %u\n",k+1,(dia+1));
    }

    printf("Cadena %s\n",semana);

    return(0);
    getchar();
}

```

Compilamos y ejecutamos el programa y lo que hace es imprimir “El dia es l” luego “El dia es m”, después “El dia es i” y así seguidamente hasta acabar con todas las columnas de la matriz. Después se espera a que demos enter e imprime “*El puntero tiene la direccion: 0*”, luego vuelve a esperar a quedemos enter e imprime “La direccion 2 es 1” seguido de “La direccion 3 es 1” y así seguidamente hasta “La direccion 8 es 1”. Luego imprime la cadena de caracteres que surge de la matriz inicial.

```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  matriz6.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o matriz matriz6.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  matriz6.c  Plantillas  Vídeos
Desktop    examples.desktop  matriz    Música      Público
ubuntu@ubuntu:~$ ./matriz
El dia es l
El dia es m
El dia es i
El dia es j
El dia es v
El dia es s
El dia es d
El puntero dia contiene la direccion: 0

La direccion 2 es 1
La direccion 3 es 1
La direccion 4 es 1
La direccion 5 es 1
La direccion 6 es 1
La direccion 7 es 1
La direccion 8 es 1
Cadena lmjvsd
ubuntu@ubuntu:~$

```


Matrices II

Ejercicio 1. Escriba el programa y compílelo.

Este ejercicio saca cuadrados de números enteros mediante la función `pow`, pero sin llamar a la librería `math.h` sino definiendo a la propia función.

Primeramente, se declara (fuera del `main`) a la variable `pow2` (que será nuestra función) como `pow2(int)` y a la variable `dato`. Después se abre el `main` y se declaran las variables enteras `result` y nuevamente a la variable `dato`. Esto tiene una explicación, la primera vez que declaramos a la variable entera `dato` lo hacemos fuera del `main` para poder definir a nuestra función `pow2(int)` posteriormente, la segunda vez que se declara la variable entera `dato`, se hace dentro del `main` para realizar la operación que queremos realizar.

Después utilizamos la función `printf` para imprimir en pantalla “Introduce un numero entero” y mediante la función `scanf` recogemos algún número entero introducidos en el teclado y la guardamos en la variable `dato` (el que esta dentro del `main`) y definimos la variable entera `result` como `result=pow2(dato)`, es decir, la variable `result` queda definida como la variable `dato` introducida a la función `pow2(dato)` la cual saca el cuadrado del número introducido en la variable `dato`, aunque todavía no hemos definido la función `pow2(dato)`. Y mediante la función `printf` imprimimos el resultado final, es decir, el cuadrado del numero introducido.

Posteriormente, se cierra el `main`. Y ahora definimos la función `pow2(int)` y esto lo hacemos tomando la función declarada y dándole un argumento. En este caso tomamos `int pow2(int dato)` luego abrimos el paréntesis para agregar un argumento y colocamos `return` y la operación a realizar, es decir, cada vez que se llame a la función `pow2(dato)` donde `dato` es un número entero, que devuelve la multiplicación de ese numero por si mismo, es decir, eleva ese numero entero al cuadrado, y así termina nuestra función.

El código queda así:

```
#include <stdio.h>

int pow2(int);
int dato=0;

int main(void){

    int result=0;
    int dato=0;

    printf("Introduce un entero: ");
    scanf("%i",&dato);
    result=pow2(dato);
    printf("El resultado es: %i",result);

    return (0);
}

int pow2(int dato){
    return(dato*dato);
}
```

Compilamos y ejecutamos el programa y lo que hace es pedirte un número entero y devolverte su cuadrado.

```
ubuntu@ubuntu:~$ ls
cuadr.c  Desktop  examples.desktop  Música  Público
Descargas  Documentos  Imágenes  Plantillas  Videos
ubuntu@ubuntu:~$ gcc cuadr.c
ubuntu@ubuntu:~$ ls
a.out  Descargas  Documentos  Imágenes  Plantillas  Videos
cuadr.c  Desktop  examples.desktop  Música  Público
ubuntu@ubuntu:~$ ./a.out
Introduce un entero: 5
El resultado es: 25
ubuntu@ubuntu:~$ ./a.out
Introduce un entero: 4
El resultado es: 16
ubuntu@ubuntu:~$ ./a.out
Introduce un entero: 25
El resultado es: 625
ubuntu@ubuntu:~$
```

Este programa también puede hacerse mediante la función “*pow*” de la librería “*math*” que puede sacar cuadrados de números enteros. Para hacer el código del programa, además de importar la librería “*stdio.h*” también debemos importar la librería “*math.h*”.

Bien, empezamos abriendo nuestro *main* y definimos las variables enteras “*dato*” y “*result*” y las definimos, en principio, como 0. Después, mediante la función *printf* pedimos que se introduzca un número entero y mediante *scanf* recogemos el valor ingresado por el teclado y lo guardamos en la variable *dato*. Después definimos la variable *result* como “*result=pow(dato,2)*”, nótese que aquí ocupamos la función *pow* la cual tiene la siguiente sintaxis:

pow(entero, exponente)

por lo que, en nuestro código *pow* queda de la siguiente forma:

pow(dato,2)

es decir, que a al valor que tiene la variable *dato* se elevará al exponente 2 (al cuadrado).

Después mediante *printf* imprimimos en pantalla el resultado y cerramos el código con *return 0* y cerrando las llaves del *main*.

El código queda así:

```
//Este codigo saca el cuadrado de un numero con la funcion pow

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

int main(void){

    int dato=0;
    int result=0;

    printf("Introduce un entero: ");
    scanf("%i",&dato);
    result=pow(dato,2);

    printf("El cuadrado de %i es %i.\n",dato, result);

    return(0);

}
```

Compilamos y ejecutamos el programa y lo que hace es pedirte que introduzcas un número entero y después imprime la frase “El cuadrado de x es y” donde “x” es el numero entero que nosotros ingresamos y “y” es el cuadrado de ese número introducido. En mi ejemplo lo hice con los números

5, 8 y 25. y tal y como se mencionó antes, el programa nos imprime el cuadrado de dichos números respectivamente.

```
ubuntu@ubuntu:~$ ls
A1.c      Desktop  examples.desktop  Música  Público
Descargas Documentos  Imágenes          Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o A A1.c -ln
ubuntu@ubuntu:~$ ls
A      Descargas  Documentos  Imágenes  Plantillas  Vídeos
A1.c   Desktop    examples.desktop  Música    Público
ubuntu@ubuntu:~$ ./A
Introduce un entero: 5
El cuadrado de 5 es 25.
ubuntu@ubuntu:~$ ./A
Introduce un entero: 8
El cuadrado de 8 es 64.
ubuntu@ubuntu:~$ ./A
Introduce un entero: 25
El cuadrado de 25 es 625.
ubuntu@ubuntu:~$
```

Como una observación debo mencionar que por defecto no se enlaza la biblioteca *math.h* a la hora de compilar el programa y hacer el ejecutable final con *gcc*. Para solucionarlo se debe enlazar dicha biblioteca explícitamente mediante la opción *-lm* bajo la siguiente sintaxis:

```
gcc nombre_archivo.c -lm
```

En este caso, utilizamos la siguiente sintaxis porque utilice la opción *-o* para cambiarle el nombre determinado del ejecutable final:

```
gcc -o nombre_nuevo nombre_archivo.c -lm
```

Ejercicio 2. Haga el programa y corralo. (Suma de matrices)

Este ejercicio se veía un poco complicado pero es relativamente sencillo. El programa realiza una suma de matrices definiendo las funciones para sumar matrices.

Primero se declaran fuera del *main* las variables vacías *suma_mat* e *imprime_mat*. La variable *suma_mat* lleva dentro tres matrices de números enteros de 2x2, mientras que la variable *imprime_mat* lleva dentro una matriz de números enteros de 2x2.

Después, se abre el *main* y colocamos los parámetros “*int argc, char *argv[]*” y empezamos declarando las tres matrices que necesitamos. Empezamos declarando la matriz entera A de 2x2 donde la primera fila tiene a los números (1,2) y la segunda fila a (3,4). Después, declaramos la matriz entera B de 2x2 donde tanto la primer fila como la segunda tienen los mismos números (2,2). Y, finalmente, declaramos la matriz entera C de 2x2 pero no la definimos aún.

Después mediante la función *printf* imprimimos “*Hello world!*” y después procedemos a crear la parte del código que sumará las matrices, para ello utilizamos la función *printf* para imprimir “*El valor de la matriz A es:* ” y mediante la función *imprime_mat* que vamos a definir después fuera del *main* (pero que ya hemos declarado) imprimimos el valor de la matriz A. Y hacemos exactamente lo mismo pero ahora para la matriz B.

Ahora utilizamos la función *printf* para imprimir en pantalla “*La suma es:* ” y utilizamos la función *suma_mat* para sumar las matrices A y B y que el resultado lo guarde en la matriz C (esta función aun no la hemos definido pero ya la hemos declarado). Después mediante la función *imprime_mat* imprimimos el valor de la matriz C (que es el resultado de la suma de las matrices A y B). Terminamos el código y cerramos el *main*.

Hasta este momento tenemos el siguiente código:

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

void suma_mat(int a[2][2], int b[2][2], int c[2][2]);
void imprime_mat(int c[2][2]);

int main(int argc, char *argv[]){

    int A[2][2]={
        {1,2},
        {3,4}
    };
    int B[2][2]={
        {2,2},
        {2,2}
    };
    int C[2][2];

    printf("Hello, world!\n");

    //suma de matrices
    printf("La matriz A es: \n");
    imprime_mat(A);

    printf("La matriz B es: \n");
    imprime_mat(B);

    //imprime la matriz resultante
    printf("La suma es: \n");
    suma_mat(A, B, C);
    imprime_mat(C);
    return EXIT_SUCCESS;
}
```

Lo que hace falta es definir las funciones que hemos declarado fuera del *main*. Esto ultimo se hace de la siguiente forma:

Primero tomamos la variable *suma_mat* que hemos declarado anteriormente y abrimos un argumento (mediante paréntesis) para hacerla una función. Esta función se encargará de sumar dos matrices de 2x2 y guardar el resultado en una tercera matriz de 2x2. Para ello dentro del argumento de *suma_mat* declaramos a las variables *i*, *j* y abrimos un ciclo *for* que empezará con *i=0* y se seguirá ejecutando siempre que *i* sea menor a 2 y cada vez que se repita *i* incrementará su valor en 1. Y dentro de este ciclo abrimos otro ciclo *for* que empezará con *j=0* y se seguirá ejecutando siempre que *j* sea menor a 2 y cada vez que se repita *j* incrementará su valor en 1. Y dentro de este último ciclo *for* definimos la suma de matrices como $C[i][j]=A[i][j]+B[i][j]$ y cerramos todos los ciclos y el argumento de la función. Por lo que, cada vez que se llame a esta función se necesitaran 3 matrices de 2x2, las cuales dos de ellas tengan valores definidos para sumarse y guardar el resultado en la tercera matriz.

Ahora, tomamos la variable *imprime_mat*, que hemos declarado en el principio del programa fuera del *main*, y abrimos su argumento. Lo que queremos que haga es que cuando se le introduzca una matriz entera de 2x2 esta la imprima. Para ello, dentro de su argumento, declaramos las variables enteras *i*, *j* y abrimos un ciclo *for*, este ciclo iniciará con *i=0* y se seguirá repitiendo siempre que *i* sea menor a 2 y por cada vez que se repita el ciclo *i* incrementará su valor en 1. Y dentro de este ciclo abrimos otro ciclo *for* que empezará con *j=0* y se seguirá ejecutando siempre que *j* sea menor a 2 y cada vez que se repita *j* incrementará su valor en 1. Y dentro de este ultimo ciclo utilizamos la función *printf* para imprimir el valor de la matriz introducida. Después, cerramos todos los ciclos y del argumento de la función, por lo que nuestra función ha quedado definida. Lo que realiza la función *imprime_mat* es que cuando se le introduzca una matriz de 2x2, imprima sus valores en pantalla.

Esta ultima parte del código queda así:

```

void suma_mat(int A[2][2], int B[2][2], int C[2][2]){
    int i,j;

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            C[i][j]=A[i][j]+B[i][j];
        }
    }
}

void imprime_mat(int C[2][2]){
    int i, j;

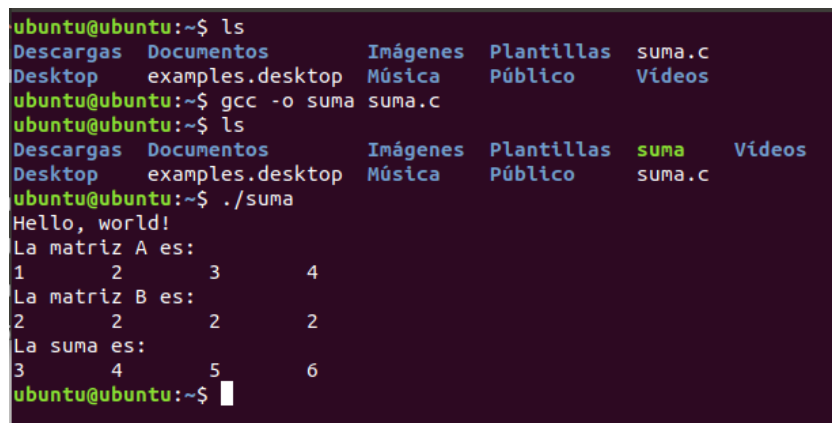
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            printf("%d\t", C[i][j]);
        }
    }
    printf("\n");
}

//fin de imprime_mat

```

Como comentario, añadir que para este código se necesita que se importen las bibliotecas *stdio.h* y *stdlib.h*.

Compilamos y ejecutamos el programa y lo primero que hace es imprimir “*Hello world!*” seguido del valor de la matriz A y después el valor de la matriz B. Después imprime “*La suma es:* ” e imprime el valor de la matriz C (la cual tiene el valor de la suma de la matriz A con la matriz B). Y, finalmente, termina el programa.



```

ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Plantillas  suma.c
Desktop    examples.desktop  Música    Público     Videos
ubuntu@ubuntu:~$ gcc -o suma suma.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Plantillas  suma  Videos
Desktop    examples.desktop  Música    Público     suma.c
ubuntu@ubuntu:~$ ./suma
Hello, world!
La matriz A es:
1      2      3      4
La matriz B es:
2      2      2      2
La suma es:
3      4      5      6
ubuntu@ubuntu:~$

```

Ejercicio 3. Realice un programa que multiplique dos matrices de 3x3.

Este ejercicio es sencillo. Lo único que debemos hacer es modificar un poco el ejercicio anterior y obtenemos el nuevo programa.

Primero exportamos las bibliotecas *stdio.h* y *stdlib.h* y se declaran, fuera del *main*, las variables vacías *multi_mat* e *imprime_mat*. La variable *multi_mat* lleva dentro tres matrices de números enteros de 3x3, mientras que la variable *imprime_mat* lleva dentro una matriz de números enteros de 3x3. Estas posteriormente las definiremos como funciones dándoles un argumento.

Después, se abre el *main* y colocamos los parámetros “*int argc, char *argv[]*” y empezamos declarando las tres matrices que necesitamos. Empezamos declarando la matriz entera A de 3x3 donde la primera fila tiene a los números (1,1,1), la segunda fila tiene a (2,2,2) y la tercer fila tiene a (3,3,3). Después, declaramos la matriz entera B de 3x3 donde tanto la primer fila como la segunda y la tercer fila tienen los mismos números (2,2,2). Y, finalmente, declaramos la matriz entera C de 3x3 pero, por el momento, no la definimos.

Después procedemos a crear la parte del código que multiplicará las matrices, para ello utilizamos la función *printf* para imprimir “El valor de la matriz A es: ” y mediante la función *imprime_mat*, que vamos a definir después fuera del *main* (pero que ya hemos declarado), imprimimos el valor de la matriz A. Y hacemos exactamente lo mismo pero ahora para la matriz B.

Ahora utilizamos la función *printf* para imprimir en pantalla “La multiplicación es: ” y utilizamos la función *multi_mat* para multiplicar las matrices A y B y que el resultado se guarde en la matriz C, (esta función aun no la hemos definido pero ya la hemos declarado). Después mediante la función *imprime_mat* imprimimos el valor de la matriz C (que es el resultado de la multiplicación de las matrices A y B). Terminamos el código y cerramos el *main*.

Esta es la primera parte del código:

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

void multi_mat(int a[3][3], int b[3][3], int c[3][3]);
void imprime_mat(int C[3][3]);

int main(int argc, char *argv[]){

    int A[3][3]={
        {1,1,1},
        {2,2,2},
        {3,3,3}
    };
    int B[3][3]={
        {2,2,2},
        {2,2,2},
        {2,2,2}
    };
    int C[3][3];

    //multiplicacion de matrices
    printf("La matriz A es: \n");
    imprime_mat(A);

    printf("La matriz B es: \n");
    imprime_mat(B);

    //imprime la matriz resultante
    printf("La multiplicacion es: \n");
    multi_mat(A, B, C);
    imprime_mat(C);
    return EXIT_SUCCESS;
}
```

Nos hace falta definir las funciones que vamos a utilizar, es decir, nos falta definir a *multi_mat* y a *imprime_mat*. Para ello haremos lo siguiente:

Primero tomamos la variable *multi_mat* que hemos declarado anteriormente y abrimos un argumento (mediante paréntesis) para hacerla una función. Esta función se encargará de multiplicar dos matrices de 3x3 y guardar el resultado en una tercera matriz de 3x3. Para ello dentro del argumento de *multi_mat* declaramos a las variables *i*, *j* y abrimos un ciclo *for* que empezará con *i=0* y se seguirá ejecutando siempre que *i* sea menor a 3 y cada vez que se repita *i* incrementará su valor en 1. Y dentro de este ciclo abrimos otro ciclo *for* que empezará con *j=0* y se seguirá ejecutando siempre que *j* sea menor a 3 y cada vez que se repita *j* incrementará su valor en 1. Y dentro de este último ciclo *for* definimos la multiplicación de matrices como:

$$C[i][j]=A[i][j]*B[i][j]$$

y cerramos todos los ciclos y el argumento de la función. Por lo que, cada vez que se llame a esta función se necesitarán 3 matrices de 3x3, las cuales dos de ellas deben tener valores definidos para multiplicarse y guardar el resultado en la tercera matriz.

Ahora, tomamos la variable *imprime_mat*, que hemos declarado en el principio del programa fuera del *main*, y abrimos su argumento. Lo que queremos que haga es que cuando se le introduzca una matriz entera de 3x3 esta la imprima. Para ello, dentro de su argumento, declaramos las variables enteras *i*, *j* y abrimos un ciclo *for*, este ciclo iniciará con *i=0* y se seguirá repitiendo siempre que *i* sea menor a 3 y por cada vez que se repita el ciclo *i* incrementará su valor en 1. Y dentro de este ciclo abrimos otro ciclo *for* que empezará con *j=0* y se seguirá ejecutando siempre que *j* sea menor a 3 y cada vez que se repita *j* incrementará su valor en 1. Y dentro de este ultimo ciclo utilizamos la función *printf* para imprimir el valor de la matriz de 3x3 introducida. Después, cerramos todos los ciclos y del argumento de la función, por lo que nuestra función ha quedado definida. Lo que realiza la función *imprime_mat* es que cuando se le introduzca una matriz de 3x3, imprima sus valores en pantalla.

Esta segunda parte del código queda así:

```
void multi_mat(int A[3][3], int B[3][3], int C[3][3]){
    int i,j;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            C[i][j]=A[i][j]*B[i][j];
        }
    }
}

void imprime_mat(int C[3][3]){
    int i, j;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("%d\t", C[i][j]);
        }
    }
    printf("\n");
}
```

Compilamos y ejecutamos el programa y lo primero que hace es imprimir el valor de la matriz A y después el valor de la matriz B. Después imprime “La suma es: ” e imprime el valor de la matriz C (la cual tiene el valor de la multiplicación de la matriz A con la matriz B). Y, finalmente, termina el programa.

```
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  Música  Público
Desktop    examples.desktop  multi.c  Plantillas  Vídeos
ubuntu@ubuntu:~$ gcc -o multi multi.c
ubuntu@ubuntu:~$ ls
Descargas  Documentos  Imágenes  multi.c  Plantillas  Vídeos
Desktop    examples.desktop  multi  Música  Público
ubuntu@ubuntu:~$ ./multi
La matriz A es:
1      1      1      2      2      2      3      3      3
La matriz B es:
2      2      2      2      2      2      2      2      2
La multiplicación es:
2      2      2      4      4      4      6      6      6
ubuntu@ubuntu:~$
```