



Ingeniería Electrónica

Informática

Apunte de Cátedra


Unidad 3: Arreglos y librerías

Francisco Murcia
Martin Morixe
Germán Tejero



Índice general

3	Arreglos y Librerías	3
3.1	Uso de Bibliotecas	3
3.2	Generación de Números Aleatorios	4
3.3	Operaciones Matemáticas	6
3.4	Vectores	8
3.4.1	Acceso a variables	8
3.4.2	Inicialización	10
3.4.3	Uso	12
3.4.4	Frecuencia de un número	13
3.4.5	Vectores paralelos	16
3.4.6	Máximo, mínimos, contadores y acumuladores	17
3.5	Cadena de caracteres	18
3.5.1	Impresión en pantalla	19
3.5.2	Ingreso por teclado	20
3.5.3	Biblioteca de funciones string.h	22
3.5.4	Funciones de conversión de tipos	25
3.5.5	Biblioteca ctype.h	26
3.6	Matrices	26
3.6.1	Acceso a variables	27
3.6.2	Inicialización	27
3.6.3	Uso	28
3.7	Guía Práctica	31
3.7.1	Uso de la librería math	31
3.7.2	Arreglos o Vectores (arrays)	31
3.7.3	Cadena de Caracteres	32
3.7.4	Matrices	32
3.7.5	Ejercicios varios de manejo de cadena de caracteres, vectores y matrices	33



3. Arreglos y Librerías

3.1 Uso de Bibliotecas

Las sentencias básicas del lenguaje C que ya vimos son:

Tipo	Sentencia
Declaración de variables	int, char, long, float, double
Programa	main, return
Condicionales	if, else, switch, case, break
Bucles	for, while, do, break, continue

Cuadro 3.1: Sentencias

El lenguaje C ofrece, además de las sentencias que ya vimos, agregar nuevos conjuntos de sentencias a las que llamamos funciones. Estas nuevas funciones las agregamos en nuestro programa para poder utilizarlas utilizando la directiva `#include`. ¿Resulta conocido?

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

Ejemplo de código 3.1: Inclusión de librerías

La directiva anterior incluye la biblioteca `stdio.h` (standard input/output) y nos da, entre otras cosas, algunas funciones que ya vimos y estuvimos usando:

```
printf  
scanf
```

Ejemplo de código 3.2: Funciones usadas

El lenguaje C ofrece una lista extensa de funciones para realizar diversas operaciones. Estas funciones están agrupadas en bibliotecas. Para ver la lista de bibliotecas y las funciones que ofrecen ver: <http://www.cplusplus.com/reference/clibrary/>.

Al finalizar esta Unidad el alumno debe poder utilizar el listado de funciones indicado a continuación. Muchas de estas funciones las veremos en esta Unidad, y otras quedarán para que el alumno las vea en el link indicado anteriormente.

stdio.h

```
scanf  
printf
```

ctype.h


```

isblank
isspace
isupper
islower
isalpha
isdigit
isxdigit
isalnum
ispunct
isgraph
isprint

```

stdlib.h

```

int rand ();
void srand(int seed);
int abs (int n);
double atof (char *s);
int atoi (char *s);
math.h
double pow(double a, double n);
double sqrt(double n);
double exp(double n);
double log(double n);
double log10(double n);
double fabs(double n);
double ceil(double n);
double floor(double n);
double sin(double n);
double cos(double n);
double tan(double n);
double acos(double n);
double asin(double n);
double atan(double n);
double sinh(double n);
double cosh(double n);
double tanh(double n);
double asinh(double n);
double acosh(double n);
double atanh(double n);

```

string.h

```

int strlen(char cadena[])
int strcpy(char destino[], char origen[])
int strncpy(char destino[], char origen[], int n)
int strcat(char destino[], char origen[])
int strncat(char destino[], char origen[], int n)
int strcmp(char cadena1[], char cadena2[])
int strncmp(char cadena1[], char cadena2[], int n)
intstrup(char cadena)
intstrlwr(char cadena)
int atoi(char cadena[])
double atof(char cadena[])

```

3.2 Generación de Números Aleatorios

Para la generación de números aleatorios utilizamos la biblioteca `stdlib.h` y `time.h`. Algunas de las funciones que ofrece `stdlib.h` son:

- `int rand ()`; retorna un número al azar entre 0 y la constante `RAND_MAX`
- `void srand(int seed)`; inicializa el generador de números aleatorios
- `int abs (int n)`; retorna el valor absoluto de `n` (entero)

- long labs(long n); retorna el valor absoluto de n (long)
- double atof (char *s); convierte la cadena s a un número real double
- int atoi (char *s); convierte la cadena s a un número entero int
- long atol (char* s); convierte la cadena s a un número entero long

Ejemplo: mostrar números al azar entre 0 y un número muy grande RAND_MAX.

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

int main()
{
    int i;

    printf("Numeros al azar:\n");
    for(i = 0; i < 10; i++)
    {
        printf("%d ", rand());
    }

    return 0;
}
```

Ejemplo de código 3.3: Número aleatorios

Ejemplo: mostrar números al azar entre 0 y un número muy grande RAND_MAX pero que no se repitan entre distintas ejecuciones del mismo programa.

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

int main()
{
    int i;

    srand(time(NULL));

    printf("Numeros al azar:\n");
    for(i = 0; i < 10; i++)
    {
        printf("%d ", rand());
    }

    return 0;
}
```

Ejemplo de código 3.4: Aleatorios entre 0 y RAND_MAX

Ejemplo: imprimir números al azar entre 0 y 9, entre 1 y 10 y entre 20 y 30:

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

int main()
{
    int i;

    srand(time(NULL));

    printf("Numeros al azar entre 0 y 9\n");
    for(i = 0; i < 40; i++)
    {
```

```
    printf("%d ", rand() % 10);  
}  
  
printf("\n\nNumeros al azar entre 1 y 10\n");  
for(i = 0; i < 40; i++)  
{  
    printf("%d ", rand() % 10 + 1);  
}  
  
printf("\n\nNumeros al azar entre 20 y 30\n");  
for(i = 0; i < 40; i++)  
{  
    printf("%d ", rand() % 11 + 20);  
}  
  
return 0;  
}
```

Ejemplo de código 3.5: Aleatorios entre rangos

3.3 Operaciones Matemáticas

El lenguaje C ofrece diversas funciones para el cálculo matemático en la biblioteca math.h. Algunas de estas son:

- double pow(double a, double n); retorna la potencia de an
- double sqrt(double n); retorna la raíz cuadrada de n
- double exp(double n); retorna el exponencial de n (en)
- double log(double n); retorna el logaritmo natural de n
- double log10(double n); retorna el logaritmo en base 10 de n
- double fabs(double n); retorna el valor absoluto de n
- double ceil(double n); retorna el entero siguiente a n
- double floor(double n); retorna el entero anterior a n
- double sin(double n); retorna el seno de n
- double cos(double n); retorna el coseno de n
- double tan(double n); retorna la tangente de n
- double acos(double n); retorna el arco coseno de n
- double asin(double n); retorna el arco seno de n
- double atan(double n); retorna el arco tangente de n
- double sinh(double n); retorna el seno hiperbólico de n
- double cosh(double n); retorna el coseno hiperbólico de n
- double tanh(double n); retorna la tangente hiperbólica de n
- double asinh(double n); retorna el arco seno hiperbólico de n
- double acosh(double n); retorna el arco coseno hiperbólico de n
- double atanh(double n); retorna el arco tangente hiperbólico de n

Esta biblioteca también define varias constantes matemáticas. Para utilizarlas es necesario predefinir a constante:

```
#define _USE_MATH_DEFINES
```

Ejemplo de código 3.6: Habilitar constantes matemáticas

Algunas de estas constantes son:

- M_E la base del logaritmo natural (e)
- M_LOG2E el logaritmo en base 2 del número e
- M_LOG10E el logaritmo en base 10 del número e
- M_LN2 el logaritmo natural del número 2

- M_LN10 el logaritmo natural del número 10
- M_PI el número PI
- M_PI_2 el número $\text{PI} / 2$
- M_PI_4 el número $\text{PI} / 4$
- M_1_PI el número $1 / \text{PI}$
- M_2_PI el número $2 / \text{PI}$
- M_SQRT2 el número raíz cuadrada de 2
- M_SQRT1_2 el número raíz cuadrada de $1/2$.

Ejemplo: escriba un programa que calcule la hipotenusa de un triángulo rectángulo dados sus dos catetos:

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

int main()
{
    float hipotenusa, cateto1, cateto2;

    printf("Ingrese el primer cateto: ");
    scanf("%f", &cateto1);

    printf("Ingrese el segundo cateto: ");
    scanf("%f", &cateto2);

    hipotenusa = sqrt(pow(cateto1, 2) + pow(cateto2, 2));
    printf("La hipotenusa es: %.2f", hipotenusa);

    return 0;
}
```

Ejemplo de código 3.7: Ejemplo de cálculo de hipotenusa

Ejemplo: escriba un programa que escriba los valores seno(x) y coseno(x) para valores de x entre 0 y PI, con incrementos de 0,25.

```
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main()
{
    float x;
    float seno, coseno;

    for(x = 0; x < M_PI; x += 0.25)
    {
        seno = sin(x);
        coseno = cos(x);
        printf("%.2f:\t%.2f\t%.2f\n", x, seno, coseno);
    }

    return 0;
}
```

Ejemplo de código 3.8: Ejemplo de seno y coseno

3.4 Vectores

¿Como haríamos un programa que pida al usuario 2 números enteros y luego los muestre en el orden inverso al ingresado? Una posibilidad sería:

```
#include <stdio.h>

int main()
{
    int a, b;

    printf("Ingrese 2 numeros: ");
    scanf("%d %d", &a, &b);

    printf("%d %d\n", b, a);

    return 0;
}
```

Ejemplo de código 3.9: Ingreso de dos números

¿Y si ahora en vez de dos números queremos que el usuario pueda ingresar 10 números? Lo solucionaríamos con 10 variables. ¿Y si fueran 100 números? Esta solución vemos que se torna inviable.

Para poder almacenar muchas variables del mismo tipo utilizando un único nombre de variable existen los vectores. Estas variables están en la computadora de forma contigua, una al lado de la otra.

Para declarar un vector de 5 variables enteras:

```
int a[5];
```

Para declarar un vector de 4 variables reales:

```
float b[4];
```

Entonces la variable a se vería así:

a				
a[0]	a[1]	a[2]	a[3]	a[4]

La variable a tiene internamente espacio para cinco variables, siendo la primera a[0], la segunda a[1], y la última a[4]. En estas cinco variables se pueden guardar números enteros (int).

Lo mismo sucede con el vector b:

b			
b[0]	b[1]	b[2]	b[3]

La variable b tiene internamente espacio para cuatro variables, siendo la primera b[0], la segunda b[1], y la última b[3]. En estas cuatro variables se pueden guardar números reales (float).

3.4.1 Acceso a variables

Para asignar un número a un vector, utilizamos el nombre de la variable (nombre del vector) y el índice al que queremos acceder. Por ejemplo, para guardar el valor 1 al primer elemento del vector y el valor 2 al segundo:

```
a[0] = 1;
a[1] = 2;
```

Ejemplo de código 3.10: Asignación a elementos

a	1	2			
	a[0]	a[1]	a[2]	a[3]	a[4]

Para leer el cuarto elemento de un vector y almacenarlo en x (siendo x otra variable int):

```
x = a[3];
```

Ejemplo de código 3.11: Acceso a elementos

Para sumar los primeros 2 elementos y guardarlos en la variable x:

```
x = a[0] + a[1];
```

Ejemplo de código 3.12: Suma de elementos

Ahora que sabemos trabajar con vectores, podemos resolver el ejercicio propuesto inicialmente (para simplificar el ingreso de datos usaremos 5 elementos únicamente).

```
#include <stdio.h>

#define N 5

int main()
{
    int a[N];
    int i;

    /* Inicializo vector. */
    for(i = 0; i < N; i++)
    {
        printf("Ingrese un numero: ");
        scanf("%d", &a[i]);
    }

    /* Muestro vector al inverso. */
    printf("Al inverso: ");
    for(i = N - 1; i >= 0; i--)
    {
        printf("%d ", a[i]);
    }
    printf("\n\n");

    return 0;
}
```

Ejemplo de código 3.13: Ejemplo con arreglo de 5 elementos

Y podemos modificar el ejercicio para inicializar un vector de 20 elementos con números al azar entre 0 y 9, y mostrarlos en el orden en que aparecieron y en el orden inverso:

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

#define N 20

int main()
{
    int a[N];
    int i;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));
```

```
/* Inicializo vector con nros al azar entre 0 y 9. */
for(i = 0; i < N; i++)
{
    a[i] = rand() % 10;
}

/* Muestro el vector original. */
printf("Al original:\n");
for(i = 0; i < N; i++)
{
    printf("%d ", a[i]);
}

/* Muestro el vector al inverso. */
printf("\n\nAl inverso:\n");
for(i = N - 1; i >= 0; i--)
{
    printf("%d ", a[i]);
}
printf("\n\n");

return 0;
}
```

Ejemplo de código 3.14: Elementos al azar

3.4.2 Inicialización

Antes de utilizar un vector es necesario inicializarlo. Esta inicialización consiste en asignar a cada posición del vector los valores iniciales.

Hay dos maneras de inicializar un vector:

- De forma dinámica
- De forma estática

Diremos que un vector se inicializa de forma estática cuando en el programa, en el momento de declarar un vector, se especifican sus valores. Por el contrario, diremos que un vector se inicializa de forma dinámica cuando el vector toma sus primeros valores en el momento en que se ejecuta el programa (en contraposición a cuando se declara).

3.4.2.1 Inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con todos sus valores en cero.

```
int main()
{
    int vec[N];
    int i;
    for( i = 0; i < N; i++)
    {
        vec[i] = 0;
    }
}
```

Ejemplo de código 3.15: Inicialización dinámica en cero

El siguiente fragmento de código inicializa un vector dinámicamente con números consecutivos comenzando de 0 (0, 1, 2, ...):

```
int main()
{
    int vec[N];
```

```
int i;
for( i = 0; i < N; i++)
{
    vec[i] = i;
}
```

Ejemplo de código 3.16: Inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con números consecutivos comenzando de 1 (1, 2, 3, ...):

```
int main()
{
    int vec[N];
    int i;
    for( i = 0; i < N; i++)
    {
        vec[i] = i + 1;
    }
}
```

Ejemplo de código 3.17: Otro ejemplo de inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con números pares consecutivos:

```
int main()
{
    int vec[N];
    int i;
    for( i = 0; i < N; i++)
    {
        vec[i] = 2 + i * 2;
    }
}
```

Ejemplo de código 3.18: Más ejemplos de inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con todos sus valores en 10.

```
int main()
{
    int vec[N];
    int i;
    for( i = 0; i < N; i++)
    {
        vec[i] = 10;
    }
}
```

Ejemplo de código 3.19: Inicialización dinámica en 10

3.4.2.2 Inicialización estática

El siguiente fragmento de código inicializa estáticamente los 3 elementos del vector con los valores 0, 1 y 2 respectivamente.

```
int main()
{
    int vec[3] = { 0, 1, 2 };
```

Ejemplo de código 3.20: Inicialización estática

El siguiente fragmento de código inicializa estáticamente los 3 elementos del vector con números pares consecutivos.

```
int main()
{
    int vec[3] = { 2, 4, 6 };
```

Ejemplo de código 3.21: Otro ejemplo de inicialización estática

El siguiente fragmento de código inicializa estáticamente un vector con su primer elemento en 1, y todos los elementos restantes en 0 (inicialización implícita).

```
int main()
{
    int vec[10] = { 1 };
```

Ejemplo de código 3.22: Inicialización estática de todos los elementos a 1

Nota: esta inicialización utiliza una propiedad de los segmentos de programa. Esto se verá con sumo detalle en la asignatura Sistemas Operativos

El siguiente fragmento de código inicializa estáticamente un vector con su primer elemento en 1, el siguiente en 2 y todos los elementos restantes en 0 (inicialización implícita).

```
int main()
{
    int vec[10] = { 1, 2 };
```

Ejemplo de código 3.23: Inicialización estática repitiendo

Nota: esta inicialización utiliza una propiedad de los segmentos de programa. Esto se verá con sumo detalle en la asignatura Sistemas Operativos

El siguiente fragmento de código inicializa estáticamente los dos vectores con todos sus elementos en 0. Ambas inicializaciones tienen el mismo efecto

```
int main()
{
    int vec1[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    int vec2[10] = { 0 };
```

Ejemplo de código 3.24: Inicialización estática en cero

Nota: esta inicialización utiliza una propiedad de los segmentos de programa. Esto se verá con sumo detalle en la asignatura Sistemas Operativos

Nota: cuando un vector se inicializa estáticamente podemos no poner en su declaración la cantidad máxima de elementos (valor que está entre corchetes). De esta manera el vector tendrá una cantidad de elementos igual a la cantidad declarada. Por ejemplo:

```
int v[] = { 0, 1, 2, 3 };
```

Ejemplo de código 3.25: Inicialización estática sin indicar tamaño

tendrá una capacidad de 4 elementos.

3.4.3 Uso

Ejemplo: inicializar un vector con 10 números al azar y mostrar la suma de todos sus elementos.

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

#define N 10
```

```
int main()
{
    int a[N], i, suma;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));

    /* Inicializa el vector. */
    for(i = 0; i < N; i++)
    {
        a[i] = rand() % 10;
    }

    /* Calculo su suma. */
    suma = 0;
    for(i = 0; i < N; i++)
    {
        suma = suma + a[i];
    }

    /* Muestro el resultado. */
    printf("La suma del vector es: %d\n", suma);

    return 0;
}
```

Ejemplo de código 3.26: Ejemplo de uso de arreglos

Ejercitar 3.2 — Hacer los ejercicios de la sección 3.7.2 Arreglos o Vectores (arrays), hasta el ejercicio 9. ■

3.4.4 Frecuencia de un número

La frecuencia de un número identifica la cantidad de veces que aparece dicho número en una serie de números.

Ejemplo: obtener 10 números al azar entre 0 y 4 y mostrar cuantas veces se obtuvo cada número (frecuencia). En este caso, necesitamos obtener la frecuencia de 5 números (del 0 al 4).

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

#define MUESTRAS      10 /* cantidad de muestras */
#define CANTIDAD_NROS  5 /* numeros entre 0 y 4 ==> 5 posiciones */

int main()
{
    int frecuencia[CANTIDAD_NROS] = { 0 };
    int i, numero;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));

    /* Proceso numeros. */
    for(i = 0; i < MUESTRAS; i++)
    {
        numero = rand() % 5; /* numeros entre 0 y 4. */
        frecuencia[numero]++;
    }

    /* Muestro resultado. */
}
```



```
for(i = 0; i < CANTIDAD_NROS; i++)
{
    printf("%d salio %d veces\n", i, frecuencia[i]);
}

return 0;
}
```

Ejemplo de código 3.27: Some Code

Ejemplo: obtener 10 numeros al azar entre 1 y 5 y mostrar cuantas veces se obtuvo cada número (frecuencia). En amarillo se muestran los cambios con respecto al ejercicio anterior.

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

#define MUESTRAS      10  /* cantidad de muestras. */
#define CANTIDAD_NROS 5   /* numeros entre 0 y 4 ==> 5 posiciones. */

int main()
{
    int frecuencia[CANTIDAD_NROS] = { 0 };
    int i, numero;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));

    /* Proceso numeros. */
    for(i = 0; i < MUESTRAS; i++)
    {
        numero = rand() % 5 + 1; /* numeros entre 1 y 5. */
        frecuencia[numero - 1]++;
    }

    /* Muestro resultado. */
    for(i = 0; i < CANTIDAD_NROS; i++)
    {
        printf("%d salio %d veces\n", i + 1, frecuencia[i]);
    }

    return 0;
}
```

Ejemplo de código 3.28: Otro ejemplo de cálculo de frecuencia

Ejemplo: otra forma de resolver el ejercicio anterior utilizando el switch.

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

#define MUESTRAS      10  /* cantidad de muestras. */
#define CANTIDAD_NROS 5   /* numeros entre 0 y 4 == 5 posiciones. */

int main()
{
    int frecuencia[CANTIDAD_NROS] = { 0 };
    int i, numero;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));
```

```

/* Proceso numeros. */
for(i = 0; i < MUESTRAS; i++)
{
    numero = rand() % 5 + 1; /* numeros entre 1 y 5. */
    switch(numero)
    {
        case 1:
            frecuencia[0]++;
            break;
        case 2:
            frecuencia[1]++;
            break;
        case 3:
            frecuencia[2]++;
            break;
        case 4:
            frecuencia[3]++;
            break;
        case 5:
            frecuencia[4]++;
            break;
        default:
            printf("Error!");
            break;
    }
}

/* Muestro resultado. */
for(i = 0; i < CANTIDAD_NROS; i++)
{
    printf("%d salio %d veces\n", i + 1, frecuencia[i]);
}

return 0;
}

```

Ejemplo de código 3.29: Ejemplo de cálculo de la frecuencia de un número

¿Puede determinar que versión es mejor? ¿Por qué? Ayuda: ¿cuál es más sencillo de leer y entender? si cambian la cantidad de muestras y/o rango de números, ¿cuál será más sencillo de modificar?

Ejemplo: obtener 10 numeros al azar pares 2, 4, 6 y 8 y mostrar cuantas veces se obtuvo cada número (frecuencia).

```

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

#define MUESTRAS      10  /* cantidad de muestras */
#define CANTIDAD_NROS 4

int main()
{
    int frecuencia[CANTIDAD_NROS] = { 0 };
    int i, numero;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));

    /* Proceso numeros */
    for(i = 0; i < MUESTRAS; i++)
    {
        numero = rand() % 4 + 1; /* numeros entre 1 y 4. */
    }
}

```

```

    numero *= 2; /* obtiene el numero par 2, 4, 6 u 8. */

    printf("%d ", numero);
    switch(numero)
    {
        case 2:
            frecuencia[0]++;
            break;
        case 4:
            frecuencia[1]++;
            break;
        case 6:
            frecuencia[2]++;
            break;
        case 8:
            frecuencia[3]++;
            break;
        default:
            printf("Error!");
            break;
    }
}

/* Muestro resultado. */
printf("\n\n");
for(i = 0; i < CANT_NUMEROS; i++)
{
    printf("%d salio %d veces\n", 2 + i * 2, frecuencia[i]);
}

return 0;
}

```

Ejemplo de código 3.30: Cálculo de frecuencia utilizando switch

Podemos utilizar otros ejemplos con inicialización de vectores con vocales, consonantes, números impares, números pares, valores posibles de un dado, etc:

```

int vocales[] = { 'a', 'e', 'i', 'o', 'u' };
int pares[] = { 2, 4, 6, 8, 10 };
int impares[] = { 1, 3, 5, 7, 9 };
int dado[] = { 1, 2, 3, 4, 5 };

```

Ejemplo de código 3.31: Arreglos estáticos

Ejercitar 3.3 — Hacer los ejercicios de la sección 3.7.2 Arreglos o Vectores (arrays), del ejercicio 10 al 16. ■

3.4.5 Vectores paralelos

Como vimos, un vector almacena datos de un mismo tipo. Pero muchas veces sucede que queremos hacer operaciones con cosas relacionadas que no son del mismo tipo. Por ejemplo, podríamos querer registrar la temperatura de distintas ciudades. Las ciudades serán nombradas con una letra y las temperaturas con un número real.

En este caso necesitaríamos 2 vectores (una para la ciudad y otro para la temperatura) de manera que un índice del vector ciudad indicará el nombre de la ciudad, mientras que el mismo índice en el vector temperatura, tendrá su temperatura asociada.

```

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

```

```

#include <time.h>
#include <ctype.h>

#define N    10  /* Maxima cantidad de ciudades a procesar. */

int main()
{
    char ciudad[N];
    float temp[N];
    int cantidad, i;

    /* Inicializo generador de numeros aleatorios. */
    srand(time(NULL));

    /* Procesa numeros */
    cantidad = 0;
    for(i = 0; i < N; i++)
    {
        printf("Ingrese ciudad (una letra. o 1 para finalizar): ");
        scanf(" %c", &ciudad[i]); /* note el espacio entre la comilla y %c
            */

        if(ciudad[i] == '1')
        {
            break;
        }
        ciudad[i] = toupper(ciudad[i]); /* asegura que sea mayuscula. */

        printf("Ingrese temperatura para ciudad %c: ", ciudad[i]);
        scanf("%f", &temp[i]);

        cantidad++;
        printf("\n");
    }

    /* Muestra resultado. */
    printf("\n\nCiudad\tTemp\n-----\t----\n");
    for(i = 0; i < cantidad; i++)
    {
        printf("%c\t%.2f\n", ciudad[i], temp[i]);
    }

    return 0;
}

```

Ejemplo de código 3.32: Ejemplo de uso de arreglos paralelos

3.4.6 Máximo, mínimos, contadores y acumuladores

Es posible calcular máximos y mínimos de vectores. También es posible utilizar acumuladores y contadores.

- Ejemplo de contadores: contar cuántas veces está el número 3 en un vector.
- Ejemplo de acumuladores: sumar todos los elementos de un vector.
- Ejemplo de mínimos: hallar el número menor de un vector.
- Ejemplo de máximos: hallar el número máximo de un vector.

Ejercitar 3.4 — Hacer los ejercicios de la sección 3.7.2 Arreglos o Vectores (arrays), todos los ejercicios. ■

3.5 Cadena de caracteres

Cadenas de caracteres, strings, o cadenas de texto se llama al tratamiento de texto en el lenguaje C. Esto nos permite poder ingresar y procesar palabras, oraciones y frases.

Una cadena de caracteres es un vector de caracteres, que tiene como característica que su último caracter es el 0 (escrito como '0'). Este último caracter 0 es utilizado por el lenguaje C como bandera para identificar cuando una cadena de caracteres finaliza.

Por lo tanto, una cadena de caracteres tiene el mismo tratamiento que un vector de caracteres. Declaración de cadenas de caracteres:

```
char cadena1[] = { 'U', 'N', 'I', 'V', 'E', 'R', 'S', 'I', 'D', 'A', 'D',  
                  , '\0' };  
char cadena2[] = "UNIVERSIDAD";
```

Ejemplo de código 3.33: Inicialización estática de cadenas de caracteres

Ambos vectores almacenan la misma cadena de caracteres, pero su inicialización estática fue diferente. Preferimos la segunda inicialización estática porque es más sencilla de escribir y de leer. Nótese que por más que explícitamente no esté escrito el caracter 0, al estar encerrado entre comillas dobles el lenguaje C entiende que se trata de una cadena de caracteres y genera automáticamente este caracter de finalización. Aquí veremos la diferencia entre caracteres y cadenas:

- 'A' representa al caracter A. Ocupa un solo byte (un solo char)
- Ärepresenta a la cadena de caracteres A. Ocupa 2 bytes (dos chars), uno para la letra A y otro para el caracter de finalización '0'

Por lo tanto, al usar vector hay que tener en cuenta la longitud del mismo: nos debemos asegurar que el tamaño del vector tiene suficientes elementos para almacenar la cadena más un caracter más (el caracter '0').

Todo lo aprendido acerca de vectores aplica para cadenas. Por lo tanto, no es posible asignar una cadena a un vector:

```
char cadena[80];  
cadena = "HOLA!"; /* INCORRECTO!! */
```

Ejemplo de código 3.34: Ejemplo de inicialización incorrecta

Sólo es posible inicializar su valor en forma estática (como ya vimos):

```
char cadena[80] = "HOLA!"; /* CORRECTO */
```

Ejemplo de código 3.35: Ejemplo de inicialización correcta

O dinámica:

```
char cadena[80];  
cadena[0] = 'H';  
cadena[1] = 'O';  
cadena[2] = 'L';  
cadena[3] = 'A';  
cadena[4] = '!';  
cadena[5] = '\0';
```

Ejemplo de código 3.36: Inicialización dinámica

A las cadenas de caracteres podemos asignarle valores de manera más sencilla utilizando funciones como strcpy():


```
char cadena[80];
strcpy(cadena, "Hola");
```

Ejemplo de código 3.37: Copia de cadenas de caracteres

3.5.1 Impresión en pantalla

Para imprimir una cadena de caracteres por pantalla podemos utilizar el ya conocido printf con el formato de carácter (%c) tratando a la cadena como a un vector. Es decir:

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

int main()
{
    char ciudad[80];
    int i;

    strcpy(ciudad, "Bariloche");
    i = 0;
    while(ciudad[i] != '\0')
    {
        printf("%c", ciudad[i]);
        i++;
    }
    printf("\n");

    return 0;
}
```

Ejemplo de código 3.38: Impresión de cadenas

Pero C provee un método más fácil: para imprimir cadena de caracteres por pantalla utilizaremos printf con el formato %s. He aquí un ejemplo:

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

int main()
{
    char ciudad[80];

    strcpy(ciudad, "Bariloche");
    printf("Yo vivo en %s.\n", ciudad);

    return 0;
}
```

Ejemplo de código 3.39: Impresión de cadenas utilizando %s

Es posible especificar el ancho del campo en donde se deberá escribir la cadena. Si en el ejemplo anterior usamos

```
printf("Yo vivo en %30s.\n", ciudad);
```

Ejemplo de código 3.40: Ancho en la impresión

Entonces el nombre de la ciudad saldría escrito en un espacio de 30 caracteres con blancos a la izquierda.

3.5.2 Ingreso por teclado

Para el ingreso de cadena de caracteres por teclado podemos utilizar:

- scanf
- fgets

3.5.2.1 Ingreso por teclado con scanf()

Hasta ahora habíamos visto el ingreso de datos utilizando la función scanf. Aprendimos, además, que la función scanf puede leer una cadena de caracteres desde el teclado utilizando el formato %s. El inconveniente que presenta es que sólo lee caracteres del teclado hasta el primer espacio o hasta el enter ('\n').

Ejemplo:

```
#include <stdio.h>

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];

    printf("Ingrese una frase: ");
    scanf("%30s", &cadena[0]);
    printf("%s\n", cadena);

    return 0;
}
```

Ejemplo de código 3.41: Ingreso de cadenas

Nota: En el ejemplo anterior el formato es “%30s” para leer una palabra de a lo sumo 30 caracteres. No lee espacios en blanco.

Otro formato que admite scanf es el de leer todo hasta el '\n'. Para esto se utiliza el formato '[\n]'.

```
#include <stdio.h>

#define LONGITUD 80

int main()
{
    char cadena[LONGITUD];

    printf("Ingrese una frase: ");
    scanf("%[\n]", &cadena[0]);

    printf("%s\n", cadena);

    return 0;
}
```

Ejemplo de código 3.42: Ingreso usando scanf

3.5.2.2 Ingreso por teclado con fgets()

La última función para el ingreso de datos por teclado que vamos a ver es la función fgets. Ésta se utiliza para leer datos de archivos, pero también se la puede utilizar para leer datos por teclado (como vamos a ver enseguida). Las ventajas que presenta esta función son:

1. es sencilla y consume todo lo que el usuario ingresa por teclado (hasta el '\n' inclusive);

2. permite controlar la cantidad de caracteres leídos de teclado para no sobrepasar el tamaño del buffer (del tamaño de la cadena definida); y
3. agrega automáticamente el ‘
0’ al final de la cadena leída

En los sig. ejemplos se puede utilizar `fgets(cadena, LONGITUD, stdin);` o bien `fgets(&cadena[0], LONGITUD, stdin);`. Son equivalentes. Utilice el que le resulte más claro.

Ejemplo:

```
#include <stdio.h>
#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];

    printf("Ingrese una frase: ");
    fgets(cadena, LONGITUD, stdin);
    printf("%s\n", cadena);

    return 0;
}
```

Ejemplo de código 3.43: Ingreso usando fgets

Un inconveniente que presenta `fgets()` es que el ‘
n’ queda dentro de la cadena de caracteres. Para limpiar esta secuencia de caracteres inválidos podemos reemplazarlos por ‘
0’ como se muestra a continuación (nota: en el ejemplo siguiente la función `strlen()` retorna la longitud de la cadena):

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

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];
    int i;

    printf("Ingrese una frase: ");
    fgets(cadena, LONGITUD, stdin);

    /* Recorre cadena reemplazando los caracteres '\r' y '\n' a '\0' */
    for( i = 0; i < (int) strlen(cadena); i++)
    {
        if(cadena[i] == '\r' || cadena[i] == '\n')
        {
            cadena[i] = '\0';
        }
    }

    printf("%s", cadena);

    return 0;
}
```

Ejemplo de código 3.44: Limpieza en el ingreso usando fgets

Y como sabemos que esos caracteres especiales ‘
r’ y/o ‘
n’ están al final, podemos resolverlo reemplazando el texto amarillo del ejemplo anterior por el siguiente código:

```

/* Recorre de atras hacia adelante reemplazando '\r' y '\n' a '\0' */
i = (int) strlen(cadena) - 1;
while(cadena[i] == '\r' || cadena[i] == '\n')
{
    cadena[i] = '\0';
    i--;
}

```

Ejemplo de código 3.45: Limpieza de la entrada

3.5.3 Biblioteca de funciones string.h

En esta parte veremos las 3 funciones principales para el uso de cadena de caracteres provistas por la biblioteca string.h:

- `strlen(char cadena[])` retorna la cantidad de caracteres de cadena
- `strcpy(char destino[], char origen[])` copia la cadena origen en destino
- `strncpy(char destino[], char origen[], int n)` copia los primeros n caracteres de cadena origen en destino
- `strcat(char destino[], char origen[])` concatena (suma) la cadena origen en destino
- `strncat(char destino[], char origen[], int n)` concatena (suma) los primeros n caracteres de la cadena origen en destino
- `strcmp(char cadena1[], char cadena2[])` Compara cadena1 y cadena2 retornando 0 si son iguales, un valor negativo si cadena1 < cadena2, y un valor positivo si cadena1 > cadena2
- `strncmp(char cadena1[], char cadena2[], int n)` Compara los primeros n caracteres de cadena1 y cadena2 retornando 0 si son iguales, un valor negativo si cadena1 < cadena2, y un valor positivo si cadena1 > cadena2
- `strupr(char cadena)` convierte minúsculas en mayúsculas
- `strlwr(char cadena)` convierte mayúsculas en minúsculas

Función `strlen()`. Para ver la longitud real de un string o cadena de caracteres usamos `strlen`.

Ejemplo:

```

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

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];
    int longitud;

    printf("Ingrese una frase: ");
    fgets(cadena, LONGITUD, stdin); /* o %cadena[0] */
    cadena[strlen(cadena) - 1] = '\0'; /* limpia '\n' final */

    longitud = strlen(cadena); /* o strlen(%cadena[0]) */
    printf("La frase tiene %d caracteres.\n", longitud);

    return 0;
}

```

Ejemplo de código 3.46: Ejemplo del cálculo del tamaño de la cadena

Función `strcpy()`. Otra cosa que se puede hacer es copiar un string usando `strcpy`. El siguiente ejemplo pide por separado el nombre, y luego el apellido. Luego copia el nombre en otra variable, le agrega un espacio en blanco, y por último le agrega el apellido al final:

```

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

#define LONGITUD 100

int main()
{
    char nombre[LONGITUD], basura;
    char apellido[LONGITUD];
    char completo[LONGITUD]; /* nombre y apellido */
    int longNombre; /* longitud de nombre */

    printf("Ingrese su nombre: ");
    scanf("%[^\n]", nombre); /* o &nombre[0] */
    scanf("%c", &basura); /* elimina el \n para que no lo tome el sig
        scanf */

    printf("Ingrese su apellido: ");
    scanf("%[^\n]", apellido);

    /* copiamos nombre en la cadena completo */
    strcpy(completo, nombre);

    /* copiamos apellido en la variable completo,
     * pero a continuacion del nombre */
    longNombre = strlen(nombre); /* o &nombre[0] */
    completo[longNombre] = ' ';
    strcpy(&completo[longNombre + 1], apellido);

    printf("Su nombre completo es: %s.\n", completo);

    return 0;
}

```

Ejemplo de código 3.47: Ejemplo de lectura de apellido y nombre

Función `strcat()`. Esto mismo se podría haber resuelto concatenando cadena de caracteres. Es decir, dado el nombre en una variable y el apellido en otra, podríamos copiar el nombre a una tercer variable, luego concatenarle (agregarle) un espacio en blanco, y por último concatenarle el apellido. Esto es lo que hace el siguiente ejemplo (nótese que al concatenar el espacio, el mismo es agregado como cadena de caracteres, como string, entre comillas dobles y no simples):

```

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

#define LONGITUD 100

int main()
{
    char nombre[LONGITUD];
    char apellido[LONGITUD];
    char completo[LONGITUD]; /* nombre y apellido */
    char basura;

    printf("Ingrese su nombre: ");
    scanf("%[^\n]", nombre); /* o &nombre[0] */
    scanf("%c", &basura); /* elimina el \n */

    printf("Ingrese su apellido: ");
    scanf("%[^\n]", apellido); /* o &apellido[0] */

    /* copiamos nombre en la cadena completo */

```



```
strcpy(completo, nombre);

/* concatenamos un espacio en blanco */
strcat(completo, " ");

/* concatenamos el apellido */
strcat(completo, apellido);

printf("Su nombre completo es: %s.\n", completo);

return 0;
}
```

Ejemplo de código 3.48: Impresión de nombre completo

Función strcmp(). También es posible comparar dos cadenas para ver si son iguales y así poder tomar alguna decisión. Por ejemplo, ingresar una oración, palabra a palabra, y finalizar cuando se ingrese la cadena punto (".").

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

#define LONGITUD 1000

int main()
{
    char oracion[LONGITUD] = { '\0' }; /* inicializado para usar strcat */
    char palabra[LONGITUD];
    char basura;
    int bandera;

    do {
        /* Pide ingreso de datos */
        printf("Ingrese una palabra: ");
        scanf("%[^\n]", palabra);
        scanf("%c", &basura); /* elimina el \n */

        /* Verifica si se ingreso "." */
        if(strcmp(palabra, ".") != 0)
        {
            bandera = 1;
            strcat(oracion, " ");
            strcat(oracion, palabra);
        } else {
            bandera = 0;
            strcat(oracion, palabra);
        }
    } while(bandera != 0);

    printf("Ud. ha ingresado la frase:\n%s\n", oracion);

    return 0;
}
```

Ejemplo de código 3.49: Ejemplo de comparación de cadenas

Nota: Al pedir una palabra ingrese, en vez de una frase (varias palabras separadas por punto). Luego pruebe cambiar los dos scanf en amarillo y repita el proceso (en el primer scanf utilizar %s y eliminar el 2do scanf).

3.5.4 Funciones de conversión de tipos

En esta parte veremos las funciones de la biblioteca `stdlib.h` para convertir cadenas de caracteres a otros tipos de datos:

- `int atoi(char cadena[])` retorna el valor numérico entero (`int`) que representa esa cadena
- `double atof(char cadena[])` retorna el valor numérico real (`double`) que representa esa cadena

Ahora que sabemos utilizar `fgets` para leer cadena de caracteres, es necesario aprender a convertir estas cadenas de caracteres a los distintos tipos de datos. Por ejemplo, si quiere leer la edad de una persona y utilizo `gets`, lo que voy a tener es una cadena con los valores de la edad, por ejemplo "35": `char cadena[LONGITUD] = "35"`. Lo que vamos a necesitar es pasar (o convertir) esa cadena a un entero. Es decir, convertir el "35" en 35 (nótese que primero está con las comillas indicando que es una cadena de caracteres, y luego sin las comillas indicando que es un entero).

Nota: estas funciones necesitan de `#include<stdlib.h>`.

Para convertir de una cadena de caracteres a un entero se utiliza `atoi()`.

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

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];
    int edad;

    printf("Ingrese una edad: ");
    scanf("%[^\n]", cadena); /* o &cadena[0] */

    edad = atoi(cadena); /* o atoi(&cadena[0]) */
    printf("Cadena:%s. Entero:%d.\n", cadena, edad);

    return 0;
}
```

Ejemplo de código 3.50: Ejemplo de cadena a entero

Para convertir de una cadena de caracteres a un float se utiliza `atof()`.

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

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];
    float pi;

    printf("Ingrese el valor de PI: ");
    scanf("%[^\n]", cadena);

    pi = atof(cadena); /* o atof(&cadena[0]); */
    printf("Cadena:%s. Float:%f.\n", cadena, pi);

    return 0;
}
```

Ejemplo de código 3.51: Ejemplo de cadena a real

Pero que sucede si el programa esperaba un número (por ejemplo la edad de una persona) y el usuario ingresa caracteres alfanuméricos (es decir, ingresa letras en vez de números)? El programa

fallará. Entonces hay que controlar lo que el usuario ingresó por teclado antes de hacer la conversión al tipo deseado. Para esto se utilizan las siguientes funciones de ctype.h vistas en el apunte anterior:

```
int isdigit(int c);
```

Ejemplo:

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

#define LONGITUD 30

int main()
{
    char cadena[LONGITUD];
    float pi;
    int longitud;
    int i;

    printf("Ingrese el valor de PI: ");
    scanf("%[^\n]", cadena);

    longitud = strlen(cadena);
    for(i = 0; i < longitud; i++)
    {
        if( ! isdigit(cadena[i]) && cadena[i] != '.' )
        {
            printf("El valor ingresado no es un numero!\n");
            return 1;
        }
    }
    pi = atof(cadena);
    printf("Cadena:%s. Float:%f.\n", cadena, pi);

    return 0;
}
```

Ejemplo de código 3.52: Ejemplo de conversión de cadena a real

3.5.5 Biblioteca ctype.h

Revise la biblioteca ctype.h desde la web como se indicó al comienzo de este apunte.

Ejercitar 3.5 — Hacer los ejercicios de la sección 3.7.3 Cadena de Caracteres. ■

3.6 Matrices

Las matrices son vectores de dos dimensiones. Supongamos la matriz m con 3 filas y 4 columnas. Entonces podemos imaginar a esta matriz de la siguiente manera:

	columnas			
	0	1	2	3
0				
1				
2				
filas	matrix m			

La matriz m se declara en lenguaje C de la siguiente manera:

```
int m[3][4];
```

Ejemplo de código 3.53: Declaración de matrices

Es decir, tipo nombre[FILAS][COLUMNAS].

3.6.1 Acceso a variables

Para acceder a cada elemento de la matriz se utilizan índices igual que en los vectores. En el siguiente diagrama se muestra el nombre de la variable y sus índices para acceder a cada celda de la matriz.

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][0]	m[2][1]	m[2][2]	m[2][3]

Para asignar un número a una matriz, utilizamos el nombre de la variable (nombre de la matriz) y los índices fila y columna al que queremos acceder. Por ejemplo,:

```
a[0][0] = 1;
a[1][2] = 2;
```

Ejemplo de código 3.54: Asignación de elementos

Modificaré la matriz de la siguiente manera:

1			
		2	

3.6.2 Inicialización

Al igual que los vectores, las matrices se pueden inicializar de dos maneras:

- De forma dinámica
- De forma estática

3.6.2.1 Inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con todos sus valores en cero.

```
int main()
{
    int m[FILAS][COLUMNAS];
    int i, j;

    for( i = 0; i < FILAS; i++)
    {
        for( j = 0; j < COLUMNAS; j++)
        {
            m[i][j] = 0;
        }
    }
}
```

Ejemplo de código 3.55: Inicialización dinámica

El siguiente fragmento de código inicializa un vector dinámicamente con valores consecutivos por filas:

```
int main()
{
```

```
int m[FILAS][COLUMNAS];
int i, j, init = 0;

for( i = 0; i < FILAS; i++)
{
    for( j = 0; j < COLUMNAS; j++)
    {
        m[i][j] = init;
        init++;
    }
}
```

Ejemplo de código 3.56: Inicialización dinámica por filas

El siguiente fragmento de código inicializa un vector dinámicamente con valores consecutivos por columnas:

```
int main()
{
    int m[FILAS][COLUMNAS];
    int i, j, init = 0;

    for( i = 0; i < COLUMNAS; i++)
    {
        for( j = 0; j < FILAS; j++)
        {
            m[j][i] = init;
            init++;
        }
    }
}
```

Ejemplo de código 3.57: Inicialización dinámica por columnas

3.6.2.2 Inicialización estática

El siguiente fragmento de código inicializa estáticamente una matriz.

```
int main()
{
    int m[2][3] =
    {
        { 0, 1, 2, },
        { 0, 1, 2, },
    }
}
```

Ejemplo de código 3.58: Inicialización estática

El siguiente fragmento de código inicializa estáticamente una matriz en cero.

```
int main()
{
    int m[2][3] = { {0} };
```

Ejemplo de código 3.59: Inicialización estática a cero

3.6.3 Uso

Ejemplo: mostrar los valores de una matriz.

```
#include <stdio.h>

#define FILAS 2
#define COLUMNAS 3
```



```
int main()
{
    int i, j;
    int m[FILAS][COLUMNAS] =
    {
        { 1, 2, 3},
        { 4, 5, 6},
    };

    for(i = 0; i < FILAS; i++)
    {
        for(j = 0; j < COLUMNAS; j++)
        {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Ejemplo de código 3.60: Ejemplo de uso de matrices

Ejemplo: inicializar una matriz con valores al azar entre 0 y 9 y luego mostrarla.

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

#define FILAS 2
#define COLUMNAS 3

int main()
{
    int i, j;
    int m[FILAS][COLUMNAS];

    srand(time(NULL));

    /* Inicializa la matriz */
    for(i = 0; i < FILAS; i++)
    {
        for(j = 0; j < COLUMNAS; j++)
        {
            m[i][j] = rand() % 10;
        }
    }

    /* Muestra la matriz */
    for(i = 0; i < FILAS; i++)
    {
        for(j = 0; j < COLUMNAS; j++)
        {
            printf("%d ", m[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Ejemplo de código 3.61: Inicialización e impresión de una matriz

Ejemplo: calcular la suma de la diagonal principal (superior-izquierda a inferior-derecha).

```
#include <stdio.h>

#define N    3

int main()
{
    int i, suma;
    int m[N][N] =
    {
        { 1, 2, 3 },
        { 4, 5, 6 },
        { 7, 8, 9 }
    };

    /* Calcula suma diagonal superior-izq a inferior-der */
    suma = 0;
    for(i = 0; i < N; i++)
    {
        suma += m[i][i];
    }

    /* Muestra resultado */
    printf("La suma de la diagonal es: %d\n", suma);

    return 0;
}
```

Ejemplo de código 3.62: Diagonal principal

Ejemplo: calcular la suma de la diagonal secundaria (superior-derecha a inferior-izquierda).

```
#include <stdio.h>

#define N    3

int main()
{
    int i, suma;
    int m[N][N] =
    {
        { 1, 2, 3 },
        { 4, 5, 6 },
        { 7, 8, 9 }
    };

    /* Calcula suma diagonal superior-der a inferior-izq */
    suma = 0;
    for(i = 0; i < N; i++)
    {
        suma += m[i][N - i - 1];
    }

    /* Muestra resultado */
    printf("La suma de la diagonal es: %d\n", suma);

    return 0;
}
```

Ejemplo de código 3.63: Diagonal secundaria

Ejercitar 3.6 — Hacer los ejercicios de la sección 3.7.4 Matrices.

3.7 Guía Práctica

3.7.1 Uso de la librería math

1. Escriba un programa que imprima una tabla con el resultado $f(x)$, para x variando de 0 a π con intervalos de 0,5:
 - a. $f(x) = x^3$
 - b. $f(x) = \sqrt{x}$
 - c. $f(x) = \text{seno}(x)$
 - d. $f(x) = \text{arcoseno}(x)$
 - e. $f(x) = \text{coseno}(x)$
 - f. $f(x) = \text{tangente}(x)$

3.7.2 Arreglos o Vectores (arrays)

2. Escriba un programa que inicialice un vector de 100 enteros al azar. Calcule e imprima su promedio.
3. Escriba un programa que inicialice un vector de 100 enteros al azar. Calcule e imprima su mínimo y máximo.
4. Escriba un programa que inicialice un vector de 100 enteros al azar. Calcule e imprima su suma, promedio, mínimo y máximo.
5. Escriba un programa que inicialice 2 vectores de 100 enteros al azar cada uno y hallar los promedios de cada vector.
6. Procesar 100 números al azar (del 1 al 10) y guardar los impares en un vector y los pares en otro.
7. En un vector de 100 elementos guardar en las posiciones pares un número al azar par, y en las posiciones impares un número al azar impar. Mostrar la posición y el número del vector entero.
8. Cargar un vector con 50 elementos numéricos y calcular la cantidad de números negativos, positivos y ceros que se encuentran en el vector.
9. Imprimir y sumar los elementos que ocupan las posiciones impares de una tabla numérica de 50 elementos.
10. Procesar 1000 edades entre (1 y 100 años) e indicar cuál es la edad de mayor frecuencia (la que más se repite).
11. Escriba un programa que pida introducir un número de N cifras (con $N > 2$) y determine si el número ingresado es capicúa. Para esto utilice un vector.
12. Escriba un programa que inicialice un vector de mil elementos con números enteros al azar entre 1 y 10 e indique el número que salió más veces.
13. Repita el ejercicio anterior pero ahora informe el número que salió más veces y el número que salió menos veces.
14. Repita el ejercicio anterior, ahora muestre las veces que salió cada número, pero únicamente mostrar los que salieron al menos una vez.
15. Escribir un programa que devuelva el nombre del día de la semana ingresado por teclado (verificando que se ingrese un dato válido). Si se ingresa 1 se mostrará D (por domingo), si se ingresa 2 se mostrará L (lunes), si se ingresa 3 se mostrará M (martes), si se ingresa 4 se mostrará I (miércoles), ...
16. Escriba un programa que devuelva el nombre del día del mes, de la misma manera que el ejercicio anterior.
17. Escriba un programa que cree dos vectores paralelos de 10 elementos. Uno debe tener letras al azar y el otro, números al azar entre 1 y 10. Mostrar solo las letras donde su vector paralelo

tenga números mayores a 5.

18. Repetir el ejercicio anterior pero ahora mostrar las letras donde su vector paralelo sea impar.
19. Escriba un programa que cree dos vectores paralelos de 10 elementos en donde se almacenarán la edad de un animal y su peso en kg en número real. El programa deberá pedir los datos por teclado y luego mostrar: a) los datos cargados, b) la edad del animal más liviano (si hubiera más de uno mostrar el primero encontrado), c) el peso del animal mayor (si hubiera más de uno, mostrar el primero encontrado)

3.7.3 Cadena de Caracteres

20. Escriba un programa que imprima un mensaje de presentación, pregunte como se llama el usuario (nombre y apellido) y lo salude. Por ejemplo, "Bienvenido Jose Perez al Sistema de la UNRN".
21. Repita el ejercicio anterior pero que ahora también se pida el género (M/m/F/f). El programa debe saludarlo anteponiendo "Sr." o "Sra". Tenga en cuenta que al ingresar el género el usuario podrá utilizar mayúsculas o minúsculas. Valide los datos ingresados.
22. Escriba un programa que pida una frase y luego imprima. Además que indique la cantidad de caracteres ingresados.
23. Escriba un programa que pida una cadena de caracteres y la copie en otra. Imprima ambas.
24. Escriba un programa que pida una cadena de caracteres e indique cuantas veces se encuentra el carácter 'a'.
25. Escriba un programa que pida una cadena por el teclado y luego un carácter. El programa debe indicar, si el carácter está, en que posiciones de la cadena se encuentra.
26. Escriba un programa que pida una frase por teclado. El programa debe indicar la cantidad de palabras de la frase.
27. Inserción de carácter en cadena. Escriba un programa que pida por el teclado una cadena, un carácter y una posición (numero). El programa deberá insertar el carácter ingresado dentro de la cadena en la posición indicada. El programa deberá manejar condiciones de error (ej: si la posición ingresada supera el largo de la cadena).
28. Escriba un programa que pida una frase, la pase a mayúsculas, y la imprima en pantalla.
29. Escriba un programa que pida una cadena por teclado que represente un número entero. El programa debe validar que la cadena representa un número (todos los caracteres son dígitos) y luego debe asignar su valor a una variable entera utilizando atoi. Imprimir la variable entera.
30. Repetir el ejercicio anterior, pero utilizando un número real, asignarla a un double, y mostrar dicho valor. El programa debe validar que la cadena sean dígitos o punto.

3.7.4 Matrices

31. Ingresar con números al azar del (0 al 9) una matriz de 3 x 3 y mostrarla
32. Ingresar con números al azar del (0 al 9) una matriz de 3 x 5 y mostrarla.
33. Ingresar un valor n y luego un valor m, a continuación inicializar con números al azar del (0 al 9) una matriz de n x m y mostrarla. Nota: definir como constantes MAX_FILAS y MAX_COLUMNAS para declarar la matriz en memoria.
34. Ingresar con números al azar del (0 al 9) una matriz de N x M, mostrarla y hallar la suma de todos sus elementos.
35. Ingresar con números al azar del (0 al 9) una matriz de 4 x 4, mostrarla y hallar el promedio de todos sus elementos.
36. Ingresar con números al azar del (0 al 9) una matriz de 4 x 4, mostrarla y hallar el máximo y mínimo de todos sus elementos.
37. Ingresar con números al azar del (0 al 9) una matriz de 4 x 4, mostrarla y hallar el máximo y mínimo de todos sus elementos. Mostrar, además, los índices de ambos.
38. Ingresar con números al azar del (0 al 9) una matriz de 3 x 4, mostrarla y crear 3 vectores

que contengan el contenido de la matriz por fila.

39. Ingresar con números al azar del (0 al 9) una matriz de 4 x 3 y hallar la suma de cada fila almacenando el resultado en un vector paralelo de 4 elementos. Mostrar la matriz y al lado de cada fila el signo = seguido de la suma correspondiente halada en el vector.
40. Ingresar con números al azar del (0 al 9) una matriz de 3 x 4, mostrarla y crear 4 vectores que contenga el contenido de la matriz por columna.
41. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N, mostrarla y hallar la sumatoria de sus diagonales.
42. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N y hallar la sumatoria de sus bordes.
43. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N y hallar la cantidad de números impares y pares que hay en sus diagonales.
44. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N y hallar la sumatoria de los números en columnas pares.
45. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N y hallar la cantidad de números entre 4 y 7.
46. Ingresar con números al azar del (0 al 9) dos matrices M1 y M2 de 4 x 4, mostrarlas y calcular la suma (matriz resultante) en una tercer matriz.
47. Ingresar con números al azar del (0 al 9) dos matrices M1 y M2 de 4 x 4, mostrarlas y calcular su producto en una tercer matriz.
48. Ingresar con números al azar del (0 al 9) una matriz cuadrada de N x N mostrarla y hallar la transpuesta y mostrar el resultado.

3.7.5 Ejercicios varios de manejo de cadena de caracteres, vectores y matrices

49. Escriba un programa que imprima una lista de amigos guardados en una agenda (inicializados estáticamente en el programa).
50. Algoritmo para hallar los números primos: Criba de Eratóstenes
 - a. Primer paso: listar los números naturales comprendidos entre 2 y 100.
 - b. Segundo paso: Se toma el primer número (el nro 2), como número primo.
 - c. Tercer paso: Se tachan todos los múltiplos del número que se acaba de indicar como primo.
 - d. Cuarto paso: se toma el siguiente número no marcado como número primo.
 - e. Quinto paso: se repite desde el tercer paso.
 - f. Finalmente, luego de procesar los 100 números, quedan sin tachar sólo los que son primos.