

# 5

## Arreglos

Los arreglos (*arrays*) permiten almacenar vectores y matrices. Los **arreglos unidimensionales** sirven para manejar vectores y los **arreglos bidimensionales** para matrices. Sin embargo, las matrices también se pueden almacenar mediante arreglos unidimensionales y por medio de apuntadores a apuntadores, temas que se verán en el capítulo siguiente.

La palabra unidimensional no indica que se trata de vectores en espacios de dimensión uno; indica que su manejo se hace mediante un subíndice. El manejo de los arreglos bidimensionales se hace mediante dos subíndices.

### 5.1 Arreglos unidimensionales

El siguiente ejemplo muestra la definición de tres arreglos, uno de 80 elementos doble precisión, otro de 30 elementos enteros y uno de 20 elementos tipo carácter.

```
double x[80];
int factores[30];
char codSexo[20];
```

Los nombres deben cumplir con las normas para los identificadores. La primera línea indica que se han reservado 80 posiciones para números doble precisión. Estas posiciones son contiguas. Es importante recalcar que en C, a diferencia de otros lenguajes, el primer elemento es `x[0]`, el segundo es `x[1]`, el tercero es `x[2]`, y así sucesivamente; el último elemento es `x[79]`.

En `x` hay espacio reservado para 80 elementos, pero esto no obliga a trabajar con los 80; el programa puede utilizar menos de 80 elementos.

C no controla si los subíndices están fuera del rango previsto; esto es responsabilidad del programador. Por ejemplo, si en algún momento el programa debe utilizar `x[90]`, lo usa sin importar si los resultados son catastróficos.

Cuando un parámetro de una función es un arreglo, se considera implícitamente que es un parámetro por referencia. O sea, si en la función se modifica algún elemento del arreglo, entonces se modificó realmente el valor original y no una copia. Pasar un arreglo como parámetro de una función y llamar esta función es muy sencillo. Se hace como en el esquema siguiente.

```
... funcion(..., double x[], ...); // prototipo
//-----
int main(void)
{
    double v[30];
    ...
    ... funcion(..., v, ...); // llamado a la funcion
```

```

    ...
}
//-----
... funcion(..., double x[],...)// definicion de la funcion
{
    // cuerpo de la funcion
    ...
}

```

En el esquema anterior, el llamado a la función se hizo desde la función `main`. Esto no es ninguna obligación; el llamado se puede hacer desde cualquier función donde se define un arreglo o donde a su vez llega un arreglo como parámetro.

También se puede hacer el paso de un arreglo como parámetro de la siguiente manera. Es la forma más usual. Tiene involucrada la noción de apuntador que se verá en el siguiente capítulo.

```

... funcion(..., double *x, ...); // prototipo
//-----
int main(void)
{
    double v[30];
    ...
    ... funcion(..., v, ...); // llamado a la funcion
    ...
}
//-----
... funcion(..., double *x, ...)// definicion de la funcion
{
    // cuerpo de la funcion
    ...
}

```

El programa del siguiente ejemplo lee el tamaño de un vector, lee los elementos del vector, los escribe y halla el promedio. Para esto utiliza funciones. Observe la manera como un arreglo se pasa como parámetro.

```

// Arreglos unidimensionales
// Lectura y escritura de un vector y calculo del promedio
//-----
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
//-----
void lectX(double *x, int n, char c );
void escrX(double *x, int n );
double promX( double *x, int n);
//=====
int main()
{
    double v[40];
    int n;

    printf("\n Promedio de elementos de un vector.\n\n");
}

```

```

printf(" numero de elementos : ");
scanf( "%d", &n);
if( n > 40 ){
    printf("\n Numero demasiado grande\n\n");
    exit(1);
}
lectX(v, n, 'v');
printf(" v : \n");
escrX(v, n);
printf(" promedio = %lf\n", promX(v, n));
return 0;
}
//=====
void lectX(double *x, int n, char c )
{
    // lectura de los elementos de un "vector".

    int i;

    for( i = 0; i < n; i++){
        printf(" %c(%d) = ", c, i+1);
        scanf("%lf", &x[i] );
    }
}
//-----
void escrX(double *x, int n )
{
    // escritura de los elementos de un "vector".

    int i;
    int nEltosLin = 5; // numero de elementos por linea

    for( i = 0; i < n; i++){
        printf("%15.8lf", x[i]);
        if( (i+1)%nEltosLin == 0 || i == n-1) printf("\n");
    }
}
//-----
double promX( double *x, int n)
{
    // promedio de los elementos del 'vector' x

    int i;
    double s = 0.0;

    if( n <= 0 ){
        printf(" promX: n = %d inadecuado\n", n);
        return 0.0;
    }
    for( i = 0; i < n; i++) s += x[i];
    return s/n;
}

```

La función `lectX` tiene tres parámetros: el arreglo, el número de elementos y una letra. Esta letra sirve para el pequeño aviso que sale antes de la lectura de cada elemento. En el ejemplo, cuando se “llama” la función, el tercer parámetro es `'v'`; entonces en la ejecución aparecerán los avisos:

```
v(1) =
v(2) =
...
```

Observe que en el `printf` de la función `lectX` aparece `i+1`; entonces para el usuario el “vector” empieza en 1 y acaba en  $n$ . Internamente empieza en 0 y acaba en  $n - 1$ .

Es importante anotar que si durante la entrada de datos hay errores, es necesario volver a empezar para corregir. Suponga que  $n = 50$ , que el usuario ha entrado correctamente 40 datos, que en el dato cuadragésimo primero el usuario digitó mal algo y después oprimió la tecla Enter. Ya no puede corregir. Sólo le queda acabar de entrar datos o abortar el programa (parada forzada del programa desde el sistema operativo) y volver a empezar. Esto sugiere que es más seguro hacer que el programa lea los datos en un archivo. La entrada y salida con archivos se verá en un capítulo posterior.

Cuando un arreglo unidimensional es parámetro de una función, no importa que el arreglo haya sido declarado de 1000 elementos y se trabaje con 20 o que haya sido declarado de 10 y se trabaje con 10. La función es de uso general siempre y cuando se controle que no va a ser llamada para usarla con subíndices mayores que los previstos. En la siguiente sección se trata el tema de los arreglos bidimensionales. Allí, el paso de parámetros no permite que la función sea completamente general.

En el siguiente ejemplo, dado un entero  $n \geq 2$  (pero no demasiado grande), el programa imprime los factores primos. El algoritmo es muy sencillo. Se busca  $d > 1$ , el divisor más pequeño de  $n$ . Este divisor es necesariamente un primo. Se divide  $n$  por  $d$  y se continúa el proceso con el último cociente. El proceso termina cuando el cociente es 1. Si  $n = 45$ , el primer divisor es 3. El cociente es 15. El primer divisor de 15 es 3. El cociente es 5. El primer divisor de 5 es 5 y el cociente es 1.

```
// Arreglos unidimensionales
// Factores primos de un entero >= 2
//-----
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
//-----
int primerDiv( int n);
int factoresPrimos( int n, int *fp, int &nf, int nfMax);
//=====
int main()
{
    int vFactPrim[40]; // vector con los factores primos
    int n;
    int nFact;        // numero de factore primos
    int i;

    printf("\n Factores primos de un entero >= 2.\n\n");

    printf(" n = ");
    scanf( "%d", &n);
    if( factoresPrimos(n, vFactPrim, nFact, 40) ){
        for(i = 0; i < nFact; i++) printf(" %d",
            vFactPrim[i]);
        printf("\n");
    }
}
```

```

    }
    else printf(" ERROR\n");
    return 0;
}
//=====
int primerDiv( int n)
{
    // n debe ser mayor o igual a 2.
    // Calcula el primer divisor, mayor que 1, de n
    // Si n es primo, devuelve n.
    // Si hay error, devuelve 0.

    int i;

    if( n < 2 ){
        printf(" primerDiv: %d inadecuado.\n", n);
        return 0;
    }
    for( i = 2; i*i <= n; i++) if( n%i == 0 ) return i;
    return n;
}
//=====
int factoresPrimos( int n, int *fp, int &nf, int nfMax)
{
    // factores primos de n
    // devuelve 0 si hay error.
    // devuelve 1 si todo esta bien.
    // fp : vector con los factores primos
    // nf : numero de factores primos
    // nfMax : tamaño del vector fp

    int d, indic;

    if( n < 2 ){
        printf(" factoresPrimos: %d inadecuado.\n", n);
        return 0;
    }
    nf = 0;
    do{
        if( nf >= nfMax ){
            printf("factoresPrimos: demasiados factores.\n");
            return 0;
        }
        d = primerDiv(n);
        fp[nf] = d;
        nf++;
        n /= d;
    } while( n > 1);
    return 1;
}

```

## 5.2 Arreglos multidimensionales

La declaración de los arreglos bidimensionales, caso particular de los arreglos multidimensionales, se hace como en el siguiente ejemplo:

```
double a[3][4];
int pos[10][40];
char list[25][25];
```

En la primera línea se reserva espacio para  $3 \times 4 = 12$  elementos doble precisión. El primer subíndice varía entre 0 y 2, y el segundo varía entre 0 y 3. Usualmente, de manera análoga a las matrices, se dice que el primer subíndice indica la fila y el segundo subíndice indica la columna.

Un arreglo tridimensional se declararía así:

```
double c[20][30][10];
```

Los sitios para los elementos de  $a$  están contiguos en el orden fila por fila, o sea,  $a[0][0]$ ,  $a[0][1]$ ,  $a[0][2]$ ,  $a[0][3]$ ,  $a[1][0]$ ,  $a[1][1]$ ,  $a[1][2]$ ,  $a[1][3]$ ,  $a[2][0]$ ,  $a[2][1]$ ,  $a[2][2]$ ,  $a[2][3]$ .

En el siguiente ejemplo, el programa sirve para leer matrices, escribirlas y calcular el producto. Lo hace mediante la utilización de funciones que tienen como parámetros arreglos bidimensionales.

```
// prog14
// Arreglos bidimensionales
// Lectura y escritura de 2 matrices y calculo del producto
//-----
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
//-----
void lectA0(double a[][40], int m, int n, char c );
void escrA0(double a[][40], int m, int n );
int prodAB0(double a[][40], int m, int n, double b[][40],
            int p, int q, double c[][40]);
//=====
int main()
{
    double a[50][40], b[20][40], c[60][40];
    int m, n, p, q;

    printf("\n Producto de dos matrices.\n\n");

    printf(" num. de filas de A    : ");
    scanf( "%d", &m);
    printf(" num. de columnas de A : ");
    scanf( "%d", &n);
    // es necesario controlar que m, n no son muy grandes
    // ni negativos

    printf(" num. de filas de B    : ");
    scanf( "%d", &p);
    printf(" num. de columnas de B : ");
    scanf( "%d", &q);
    // es necesario controlar que p, q no son muy grandes
```

```

// ni negativos

if( n != p ){
    printf(" Producto imposible\n");
    exit(1);
}
lectAO(a, m, n, 'A');
printf(" A : \n");
escrAO(a, m, n);

lectAO(b, n, q, 'B');
printf(" B : \n");
escrAO(b, n, q);

if( prodABO(a,m,n, b,p,q, c) ){
    printf(" C : \n");
    escrAO(c, m, q);
}
else printf("\ ERROR\n");
return 0;
}
//=====
void lectAO(double a[][40], int m, int n, char c )
{
    // lectura de los elementos de una matriz.

    int i, j;

    for( i = 0; i < m; i++){
        for( j=0; j < n; j++){
            printf(" %c[%d][%d] = ", c, i+1, j+1);
            scanf("%lf", &a[i][j] );
        }
    }
}
//-----
void escrAO(double a[][40], int m, int n )
{
    // escritura de los elementos de una matriz

    int i, j;
    int nEltosLin = 5; // numero de elementos por linea

    for( i = 0; i < m; i++){
        for( j = 0; j < n; j++){
            printf("%15.8lf", a[i][j]);
            if((j+1)%nEltosLin == 0 || j==n-1)printf("\n");
        }
    }
}
//-----
int prodABO(double a[][40], int m, int n, double b[][40],

```

```

int p, int q, double c[][40])
{
    // producto de dos matrices, a mxn, b pxq
    // devuelve 1 si se puede hacer el producto
    // devuelve 0 si no se puede

    int i, j, k;
    double s;

    if(m<0||n<0||p<0||q<0 || n!= p ) return 0;
    for( i=0; i < m; i++){
        for( j=0; j < q; j++){
            s = 0.0;
            for( k=0; k<n; k++) s += a[i][k]*b[k][j];
            c[i][j] = s;
        }
    }
    return 1;
}

```

Cuando en una función un parámetro es un arreglo bidimensional, la función debe saber, en su definición, el número de columnas del arreglo bidimensional. Por eso en la definición de las funciones está `a[][40]`. Esto hace que las funciones del ejemplo sirvan únicamente para arreglos bidimensionales definidos con 40 columnas. Entonces estas funciones no son de uso general. Este inconveniente se puede resolver de dos maneras:

- Mediante apuntadores y apuntadores dobles. Este tema se verá en el siguiente capítulo.
- Almacenando las matrices en arreglos unidimensionales con la convención de que los primeros elementos del arreglo corresponden a la primera fila de la matriz, los que siguen corresponden a la segunda fila, y así sucesivamente. Esta modalidad es muy usada, tiene algunas ventajas muy importantes. Se verá con más detalle más adelante.

En resumen, los arreglos bidimensionales no son muy adecuados para pasarlos como parámetros a funciones. Su uso debería restringirse a casos en que el arreglo bidimensional se usa únicamente en la función donde se define.

En el ejemplo anterior, en la función `lectA0`, antes de la lectura del elemento `a[i][j]`, el programa escribe los valores `i+1` y `j+1`, entonces para el usuario el primer subíndice empieza en 1 y acaba en `m`; el segundo empieza en 1 y acaba en `n`.

### 5.3 Cadenas

Los arreglos unidimensionales de caracteres, además de su manejo estándar como arreglo, pueden ser utilizados como cadenas de caracteres, siempre y cuando uno de los elementos del arreglo indique el fin de la cadena. Esto se hace mediante el carácter especial

```
'\0'
```

En el ejemplo

```

// Arreglo de caracteres como tal.
#include <math.h>
#include <stdio.h>

```



```

#include <stdlib.h>
int main()
{
    char aviso[30];
    int i;

    aviso[0] = 'C';
    aviso[1] = 'o';
    aviso[2] = 'm';
    aviso[3] = 'o';
    aviso[4] = ' ';
    aviso[5] = 'e';
    aviso[6] = 's';
    aviso[7] = 't';
    aviso[8] = 'a';
    aviso[9] = '?';
    for(i=0; i<= 9; i++) printf("%c", aviso[i]);
    return 0;
}

```

el arreglo `aviso` se consideró como un simple arreglo de caracteres. El programa escribe

Como esta?

En el siguiente ejemplo, el arreglo `aviso` es (o contiene) una cadena, *string*, pues hay un fin de cadena. Para la escritura se usa el formato `%s`. El resultado es el mismo.

```

// prog15b
// Cadena de caracteres
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char aviso[30];

    aviso[0] = 'C';
    aviso[1] = 'o';
    aviso[2] = 'm';
    aviso[3] = 'o';
    aviso[4] = ' ';
    aviso[5] = 'e';
    aviso[6] = 's';
    aviso[7] = 't';
    aviso[8] = 'a';
    aviso[9] = '?';
    aviso[10] = '\0';
    printf("%s", aviso);
    return 0;
}

```

Si se modifica ligeramente de la siguiente manera:

```
char aviso[30];
```

```

aviso[0] = 'C';
aviso[1] = 'o';
aviso[2] = 'm';
aviso[3] = 'o';
aviso[4] = '\\0';
aviso[5] = 'e';
aviso[6] = 's';
aviso[7] = 't';
aviso[8] = 'a';
aviso[9] = '?';
aviso[10] = '\\0';
printf("%s", aviso);

```

entonces únicamente escribe Como, ya que encuentra el fin de cadena (el primero) después de la segunda letra o.

La lectura de cadenas de hace mediante la función `gets( )`. Su archivo de cabecera es `stdio.h`. Su único parámetro es precisamente la cadena que se desea leer.

```

char nombre[81];

printf(" Por favor, escriba su nombre : ");
gets(nombre);
printf("\n Buenos dias %s\n", nombre);

```

En C++ se puede utilizar `cin` para leer cadenas que no contengan espacios. Cuando hay un espacio, es reemplazado por fin de cadena.

```

char nombre[81];

cout<<" Por favor, escriba su nombre : ";
cin>>nombre;
cout<<endl<<" Buenos dias "<<nombre<<endl;

```

Si el usuario escribe Juanito, el programa (la parte de programa) anterior escribirá Buenos dias Juanito. Pero si el usuario escribe el nombre de dos palabras Juan Manuel, el programa escribirá Buenos dias Juan.

En C++ es posible leer cadenas de caracteres con espacios mediante `cin.getline()`. Este tema no se trata en este libro.

Para tener acceso a las funciones para el manejo de cadenas, se necesita el archivo de cabecera `string.h`. Las funciones más usuales son:

```

strcpy( , )
strcat( , )
strlen( )

```

El primer parámetro de `strcpy` (*string copy*) debe ser un arreglo de caracteres. El segundo parámetro debe ser una cadena constante o una cadena en un arreglo de caracteres. La función copia en el arreglo (primer parámetro) la cadena (el segundo parámetro). Se presentan problemas si el segundo parámetro no cabe en el primero. En un manual de referencia de C puede encontrarse información más detallada sobre estas y otras funciones relacionadas con las cadenas.

La función `strlen` (*string length*) da como resultado la longitud de la cadena sin incluir el fin de cadena.

Una cadena constante es una sucesión de caracteres delimitada por dos comillas dobles; por ejemplo: "Hola". No necesita explícitamente el signo de fin de cadena, ya que C lo coloca implícitamente. La cadena constante más sencilla es la cadena vacía: "". La cadena constante de un solo carácter es diferente del carácter. Por ejemplo, "x" es diferente de 'x'.

El programa prog15b se puede escribir más rápidamente así:

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

int main()
{
    char aviso[30];

    strcpy(aviso, "Como esta?");
    printf("%s", aviso);
    printf("\n longitud = %d\n", strlen(aviso));
    return 0;
}
```

Como era de esperarse, el programa anterior escribe `Como esta?` y en la línea siguiente `longitud = 10`. Efectivamente las diez primeras posiciones del arreglo `aviso`, de la 0 a la 9, están ocupadas. La posición 10 está ocupada con el fin de cadena. El arreglo `aviso` puede contener cadenas de longitud menor o igual a 29, pues se necesita un elemento para el signo de fin de cadena.

La función `strcat` sirve para concatenar dos cadenas. El primer parámetro de `strcat` debe ser una cadena en un arreglo de caracteres. El segundo parámetro debe ser una cadena constante o una cadena en un arreglo de caracteres. La función pega la segunda cadena a la derecha de la primera cadena. Aparecen problemas si en el primer arreglo no cabe la concatenación de la primera y la segunda cadenas. La concatenación se hace de manera limpia: la función quita el fin de cadena en el primer arreglo y pega la segunda incluyendo su fin de cadena.

```
// funcion strcat

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

int main()
{
    char nombre[41], apell[41], Nombre[81];

    printf(" Por favor, escriba su nombre : ");
    gets(nombre);
    printf(" Por favor, escriba su apellido : ");
    gets(apell);
    strcpy(Nombre, nombre);
    strcat(Nombre, " ");
    strcat(Nombre, apell);
    printf("Buenos dias %s\n", Nombre);
    return 0;
}
```

## 5.4 Inicialización de arreglos

Los arreglos pequeños se pueden inicializar de la siguiente manera:

```
double x[4] = { 1.1, 1.2, 1.3, 1.4};
```

Esto es lo mismo que escribir:

```
double x[4];

x[0] = 1.1;
x[1] = 1.2;
x[2] = 1.3;
x[3] = 1.4;
```

Si dentro de los corchetes hay menos valores que el tamaño del arreglo, generalmente C asigna 0.0 a los faltantes. El ejemplo

```
double x[4] = { 1.1, 1.2};
```

produce el mismo resultado que

```
double x[4];

x[0] = 1.1;
x[1] = 1.2;
x[2] = 0.0;
x[3] = 0.0;
```

Si no se precisa el tamaño del arreglo en una inicialización, C le asigna el tamaño dado por el número de elementos. El ejemplo

```
double x[] = { 1.1, 1.2, 1.3};
```

es equivalente a

```
double x[3];

x[0] = 1.1;
x[1] = 1.2;
x[2] = 1.3;
```

En este otro ejemplo, con una cadena en un arreglo de caracteres,

```
char saludo[] = "Buenos dias";
```

resulta lo mismo que escribir

```
char saludo[12] = {'B', 'u', 'e', 'n', 'o', 's', ' ',
                  'd', 'i', 'a', 's', '\0'};
```

o igual que escribir

```

char saludo[12];

saludo[0] = 'B';
saludo[1] = 'u';
saludo[2] = 'e';
saludo[3] = 'n';
saludo[4] = 'o';
saludo[5] = 's';
saludo[6] = ' ';
saludo[7] = 'd';
saludo[8] = 'i';
saludo[9] = 'a';
saludo[10] = 's';
saludo[11] = '\0';

```

Para arreglos bidimensionales, basta con recordar que primero están los elementos de la fila 0, enseguida los de la fila 1, y así sucesivamente. La inicialización

```
double a[2][3] = { 1.1, 1.2, 1.3, 1.4, 1.5, 1.6};
```

produce el mismo resultado que

```

double a[2][3];

a[0][0] = 1.1;
a[0][1] = 1.2;
a[0][2] = 1.3;
a[1][0] = 1.4;
a[1][1] = 1.5;
a[1][2] = 1.6;

```

La siguiente inicialización también hubiera producido el mismo resultado anterior:

```
double a[][3] = { 1.1, 1.2, 1.3, 1.4, 1.5, 1.6};
```

En el ejemplo anterior, C sabe que las filas tienen tres elementos, entonces en el arreglo `a` debe haber dos filas. En el ejemplo que sigue, C asigna ceros a lo que queda faltando hasta obtener filas completas.

```
double a[][3] = { 1.1, 1.2, 1.3, 1.4};
```

Lo anterior es equivalente a

```

double a[2][3];

a[0][0] = 1.1;
a[0][1] = 1.2;
a[0][2] = 1.3;
a[1][0] = 1.4;
a[1][1] = 0.0;
a[1][2] = 0.0;

```

En las siguientes inicializaciones hay errores. Para los arreglos bidimensionales, C necesita conocer el tamaño de las filas (el número de columnas).

```
double a[] [] = { 1.1, 1.2, 1.3, 1.4, 1.5, 1.6};
double b[2] [] = { 1.1, 1.2, 1.3, 1.4, 1.5, 1.6};
```

## Ejercicios

Para cada uno de los enunciados siguientes, defina cuáles son los datos necesarios. Elabore un programa que lea los datos, llame la función (o las funciones) que realiza los cálculos y devuelve los resultados, y finalmente, haga que el programa principal (la función `main`) muestre los resultados.

- 5.1 Intercambie los elementos de un vector: el primero pasa a la última posición y el último a la primera posición, el segundo pasa a la penúltima posición y viceversa...
- 5.2 Obtenga la expresión binaria de un entero no negativo.
- 5.3 Obtenga la expresión en base  $p$  ( $p$  entero,  $2 \leq p \leq 9$ ), de un entero no negativo.
- 5.4 Obtenga la expresión hexadecimal de un entero no negativo.
- 5.5 Averigüe si una lista de números está ordenada de menor a mayor.
- 5.6 Averigüe si una lista de números está ordenada de manera estrictamente creciente.
- 5.7 Averigüe si una lista tiene números repetidos.
- 5.8 Ordenar, de menor a mayor, los elementos de una lista.
- 5.9 Averigüe si una lista ordenada de menor a mayor tiene números repetidos.
- 5.10 Dada una lista de  $n$  números, averigüe si el número  $t$  está en la lista.
- 5.11 Dada una lista de  $n$  números, ordenada de menor a mayor, averigüe si el número  $t$  está en la lista.
- 5.12 Halle el promedio de los elementos de un vector.
- 5.13 Halle la desviación estándar de los elementos de un vector.
- 5.14 Dado un vector con componentes (coordenadas) no negativas, halle el promedio geométrico.
- 5.15 Halle la moda de los elementos de un vector.
- 5.16 Halle la mediana de los elementos de un vector.
- 5.17 Dados un vector  $x$  de  $n$  componentes y una lista,  $v_1, v_2, \dots, v_m$  estrictamente creciente, averigüe cuantos elementos de  $x$  hay en cada uno de los  $m + 1$  intervalos  $(-\infty, v_1]$ ,  $(v_1, v_2]$ ,  $(v_2, v_3]$ ,  $\dots$ ,  $(v_{m-1}, v_m]$ ,  $(v_m, \infty)$ .
- 5.18 Dado un vector de enteros positivos, halle el m.c.d.
- 5.19 Dado un vector de enteros positivos, halle el m.c.m. (mínimo común múltiplo).
- 5.20 Dado un polinomio definido por el grado  $n$  y los  $n + 1$  coeficientes, calcule el verdadero grado. Por ejemplo, si  $n = 4$  y  $p(x) = 5 + 0x + 6x^2 + 0x^3 + 0x^4$ , su verdadero grado es 2.
- 5.21 Dado un polinomio halle su derivada.
- 5.22 Dado un polinomio  $p$  y un punto  $(a, b)$  halle su antiderivada  $q$  tal que  $q(a) = b$ .
- 5.23 Dados dos polinomios (pueden ser de grado diferente), halle su suma.

- 5.24** Dados dos polinomios, halle su producto.
- 5.25** Dados dos polinomios, halle el cociente y el residuo de la división.
- 5.26** Dados  $n$  puntos en  $\mathbb{R}^2$ ,  $P_1, P_2, \dots, P_n$ . Verifique que la línea poligonal cerrada  $P_1P_2\dots P_nP_1$  sea de Jordan (no tiene “cruces”).
- 5.27** Dados  $n$  puntos en  $\mathbb{R}^2$ ,  $P_1, P_2, \dots, P_n$ , tal que la línea poligonal cerrada  $P_1P_2\dots P_nP_1$  es de Jordan, averigüe si el polígono determinado es convexo.
- 5.28** Dados  $n$  puntos en  $\mathbb{R}^2$ ,  $P_1, P_2, \dots, P_n$ , tal que la línea poligonal cerrada  $P_1P_2\dots P_nP_1$  es de Jordan, halle el área del polígono determinado.
- 5.29** Sea  $x$  un vector en  $\mathbb{R}^n$  y  $A$  una matriz de tamaño  $m \times n$  definida por una lista de  $p$  triplas de la forma  $(i_k, j_k, v_k)$  para indicar que  $a_{i_k j_k} = v_k$  y que las demás componentes de  $A$  son nulas; calcule  $Ax$ .
- 5.30** Considere un conjunto  $A$  de  $n$  elementos enteros almacenados en un arreglo  $a$ . Considere un lista de  $m$  parejas en  $A \times A$ . Esta lista define una relación sobre  $A$ . Averigüe si la lista está realmente bien definida, si la relación es simétrica, antisimétrica, reflexiva, transitiva y de equivalencia.
- 5.31** Considere un conjunto  $A$  de  $n$  elementos y una matriz  $M$  de tamaño  $n \times n$  que representa una operación binaria sobre  $A$ . Averigüe si la operación binaria está bien definida, si es conmutativa, si es asociativa, si existe elemento identidad, si existe inverso para cada elemento de  $A$ .