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 tambi´en se pueden almacenar mediante arreglos unidimensionales y por medio de apuntadores a apuntadores, temas que se ver´an en el cap´ıtulo siguiente.

La palabra unidimensional no indica que se trata de vectores en espacios de dimensi´on uno; indica que su manejo se hace mediante un sub´ındice. El manejo de los arreglos bidimensionales se hace mediante dos sub´ındices.

5.1 Arreglos unidimensionales

El siguiente ejemplo muestra la definici´on de tres arreglos, uno de 80 elementos doble precisi´on, otro de 30 elementos enteros y uno de 20 elementos tipo car´acter.

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

Los nombres deben cumplir con las normas para los identificadores. La primera l´ınea indica que se han reservado 80 posiciones para nu´meros doble precisi´on. 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 as´ı sucesivamente; el u´ltimo 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 sub´ındices est´an fuera del rango previsto; esto es responsabilidad del programador. Por ejemplo, si en algu´n momento el programa debe utilizar x[90], lo usa sin importar si los resultados son catastr´oficos.

Cuando un par´ametro de una funci´on es un arreglo, se considera impl´ıcitamente que es un par´ametro por referencia. O sea, si en la funci´on se modifica algu´n elemento del arreglo, entonces se modific´o realmente el valor original y no una copia. Pasar un arreglo como par´ametro de una funci´on y llamar esta funci´on 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 funci´on se hizo desde la funci´on main. Esto no es ninguna obligaci´on; el llamado se puede hacer desde cualquier funci´on donde se define un arreglo o donde a su vez llega un arreglo como par´ametro.

Tambi´en se puede hacer el paso de un arreglo como par´ametro de la siguiente manera. Es la forma m´as usual. Tiene involucrada la noci´on de apuntador que se ver´a en el siguiente cap´ıtulo.

```
... 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 taman˜o 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 par´ametro.

```
// 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 funci´on lectX tiene tres par´ametros: el arreglo, el nu´mero de elementos y una letra. Esta letra sirve para el pequen˜o aviso que sale antes de la lectura de cada elemento. En el ejemplo, cuando se "llama" la funci´on, el tercer par´ametro es 'v'; entonces en la ejecuci´on aparecer´an los avisos:

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

Observe que en el printf de la funci´on 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 cuadrag´esimo primero el usuario digit´o mal algo y despu´es oprimi´o la tecla Enter. Ya no puede corregir. S´olo 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 m´as seguro hacer que el programa lea los datos en un archivo. La entrada y salida con archivos se ver´a en un cap´ıtulo posterior.

Cuando un arreglo unidimensional es par´ametro de una funci´on, 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 funci´on es de uso general siempre y cuando se controle que no va a ser llamada para usarla con sub´ındices mayores que los previstos. En la siguiente secci´on se trata el tema de los arreglos bidimensionales. All´ı, el paso de par´ametros no permite que la funci´on 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 m´as pequen˜o de n. Este divisor es necesariamente un primo. Se divide n por d y se continu´a el proceso con el u´ltimo 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 : tamano 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 declaraci´on 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 l´ınea se reserva espacio para 3 × 4 = 12 elementos doble precisi´on. El primer sub´ındice var´ıa entre 0 y 2, y el segundo var´ıa entre 0 y 3. Usualmente, de manera an´aloga a las matrices, se dice que el primer sub´ındice indica la fila y el segundo sub´ındice indica la columna. Un arreglo tridimensional se declarar´ıa as´ı:

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

Los sitios para los elementos de *a* est´an 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 utilizaci´on de funciones que tienen como par´ametros 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);
      } lectA0(a, m, n, 'A'); printf(" A : \n"); escrA0(a, m, n);

      lectA0(b, n, q, 'B'); printf(" B : \n"); escrA0(b, n, q);

      if( prodAB0(a,m,n, b,p,q, c) ){ printf(" C : \n"); escrA0(c, m, q);
      }
      else printf("\ ERROR\n"); return 0 ;
}
//============================================== void lectA0(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 escrA0(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 prodAB0(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 funci´on un par´ametro es un arreglo bidimensional, la funci´on debe saber, en su definici´on, el nu´mero de columnas del arreglo bidimensional. Por eso en la definici´on de las funciones est´a a[][40]. Esto hace que las funciones del ejemplo sirvan u´nicamente 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 ver´a en el siguiente cap´ıtulo.

- Almacenando las matrices en arreglos unidimensionales con la convenci´on de que los primeros elementos del arreglo corresponden a la primera fila de la matriz, los que siguen corresponden a la
- segunda fila, y as´ı sucesivamente. Esta modalidad es muy usada, tiene algunas ventajas muy importantes. Se ver´a con m´as detalle m´as adelante.

En resumen, los arreglos bidimensionales no son muy adecuados para pasarlos como par´ametros a funciones. Su uso deber´ıa restringirse a casos en que el arreglo bidimensional se usa u´nicamente en la funci´on donde se define.

En el ejemplo anterior, en la funci´on 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 sub´ındice empieza en 1 y acaba en *m*; el segundo empieza en 1 y acaba en *n*.

5.3 Cadenas

Los arreglos unidimensionales de caracteres, adem´as de su manejo est´andar 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 car´acter 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 consider´o 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 u´nicamente escribe Como, ya que encuentra el fin de cadena (el primero) despu´es de la segunda letra o.

La lectura de cadenas de hace mediante la funci´on gets( ). Su archivo de cabecera es stdio.h. Su u´nico par´ametro 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 escribir´a Buenos dias Juanito. Pero si el usuario escribe el nombre de dos palabras Juan Manuel, el programa escribir´a 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 m´as usuales son:

```
strcpy(     ,     )
strcat(     ,     )
strlen( )
```

El primer par´ametro de strcpy (*string copy*) debe ser un arreglo de caracteres. El segundo par´ametro debe ser una cadena constante o una cadena en un arreglo de caracteres. La funci´on copia en el arreglo (primer par´ametro) la cadena (el segundo par´ametro). Se presentan problemas si el segundo par´ametro no cabe en el primero. En un manual de referencia de C puede encontrarse informaci´on m´as detallada sobre estas y otras funciones relacionadas con las cadenas.

La funci´on strlen (*string length*) da como resultado la longitud de la cadena sin incluir el fin de cadena.

Una cadena constante es una sucesi´on de caracteres delimitada por dos comillas dobles; por ejemplo: "Hola". No necesita expl´ıcitamente el signo de fin de cadena, ya que C lo coloca impl´ıcitamente. La cadena constante m´as sencilla es la cadena vac´ıa: "". La cadena constante de un solo car´acter es diferente del car´acter. Por ejemplo, "x" es diferente de 'x'.

El programa prog15b se puede escribir m´as r´apidamente as´ı:

```
#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 l´ınea siguiente longitud = 10. Efectivamente las diez primeras posiciones del arreglo aviso, de la 0 a la 9, est´an ocupadas. La posici´on 10 est´a 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 funci´on strcat sirve para concatenar dos cadenas. El primer par´ametro de strcat debe ser una cadena en un arreglo de caracteres. El segundo par´ametro debe ser una cadena constante o una cadena en un arreglo de caracteres. La funci´on pega la segunda cadena a la derecha de la primera cadena. Aparecen problemas si en el primer arreglo no cabe la concatenaci´on de la primera y la segunda cadenas. La concatenaci´on se hace de manera limpia: la funci´on 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 Inicializaci´on de arreglos

Los arreglos pequen˜os 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 taman˜o 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 taman˜o del arreglo en una inicializaci´on, C le asigna el taman˜o dado por el nu´mero 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 est´an los elementos de la fila 0, enseguida los de la fila 1, y as´ı sucesivamente. La inicializaci´on

```
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 inicializaci´on tambi´en 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 taman˜o de las filas (el nu´mero 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 cu´ales son los datos necesarios. Elabore un programa que lea los datos, llame la funci´on (o las funciones) que realiza los c´alculos y devuelve los resultados, y finalmente, haga que el programa principal (la funci´on main) muestre los resultados.

5.1 Intercambie los elementos de un vector: el primero pasa a la u´ltima posici´on y el u´ltimo a la primera posici´on, el segundo pasa a la penu´ltima posici´on y viceversa...

5.2 Obtenga la expresi´on binaria de un entero no negativo.

5.3 Obtenga la expresi´on en base p (p entero, $2 \leq p \leq 9$), de un entero no negativo.

5.4 Obtenga la expresi´on hexadecimal de un entero no negativo.

5.5 Averigu¨e si una lista de nu´meros est´a ordenada de menor a mayor.

5.6 Averigu¨e si una lista de nu´meros est´a ordenada de manera estrictamente creciente.

5.7 Averigu¨e si una lista tiene nu´meros repetidos.

5.8 Ordenar, de menor a mayor, los elementos de una lista.

5.9 Averigu¨e si una lista ordenada de menor a mayor tiene nu´meros repetidos.

5.10 Dada una lista de n nu´meros, averigu¨e si el nu´mero t est´a en la lista.

5.11 Dada una lista de n nu´meros, ordenada de menor a mayor, averigu¨e si el nu´mero t est´a en la lista.

5.12 Halle el promedio de los elementos de un vector.

5.13 Halle la desviaci´on est´andar de los elementos de un vector.

5.14 Dado un vector con componentes (coordenadas) no negativas, halle el promedio geom´etrico.

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, $v_1, v_2, ..., v_m$ estrictamente creciente, averigu¨e cuantos elementos de x hay en cada uno de los $m+1$ intervalos $(\infty, v_1]$, $(v_1, v_2]$, $(v_2, v_3]$, ..., $(v_{m-1}, v_m]$, (v_m, ∞).

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. (m´ınimo comu´n mu´ltiplo).

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 divisi´on.

5.26 Dados n puntos en $\mathbb{R}^2$, $P_1, P_2, ..., P_n$. Verifique que la l´ınea poligonal cerrada $P_1P_2...P_nP_1$ sea de Jordan (no tiene "cruces").

5.27 Dados n puntos en $\mathbb{R}^2$, $P_1, P_2, ..., P_n$, tal que la l´ınea poligonal cerrada $P_1P_2...P_nP_1$ es de Jordan, averigu¨e si el pol´ıgono determinado es convexo.

5.28 Dados n puntos en $\mathbb{R}^2$, $P_1, P_2, ..., P_n$, tal que la l´ınea poligonal cerrada $P_1P_2...P_nP_1$ es de Jordan, halle el ´area del pol´ıgono determinado.

5.29 Sea x un vector en $\mathbb{R}^n$ y A una matriz de taman˜o $m \times n$ 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 dem´as 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 relaci´on sobre A. Averigu¨e si la lista est´a realmente bien definida, si la relaci´on es sim´etrica, antisim´etrica, reflexiva, transitiva y de equivalencia.

5.31 Considere un conjunto A de n elementos y una matriz M de taman˜o $n \times n$ que representa una operaci´on binaria sobre A. Averigu¨e si la operaci´on binaria est´a bien definida, si es conmutativa, si es asociativa, si existe elemento identidad, si existe inverso para cada elemento de A.