CURSO DE PROGRAMACIÓN BÁSICA EN C/C++

(Prof. Pedro L. Recuenco Andrés)

PARTE IV

Tipo de datos ARRAY

Los *arrays* permiten agrupar datos usando un único identificador. Todos los elementos de un *array* son del mismo tipo, y para acceder a cada elemento se usan ***índices***.

Sintaxis:

Por ejemplo, si declaramos un objeto de este modo:

```
int valor;
```

El identificador 'valor' se refiere a un objeto de tipo **int**. El compilador sólo obtendrá memoria para almacenar un entero, y el programa sólo podrá almacenar y leer un único valor en ese objeto en cada momento.

Por el contrario, si declaramos un *array*:

```
int vector[10];
```

El compilador obtendrá espacio de memoria suficiente para almacenar 10 objetos de tipo *int*, y el programa podrá acceder a cada uno de esos valores para leerlos o modificarlos.

Para acceder a cada uno de los valores se usa un índice, que en este caso podrá tomar valores entre 0 y 9. Usando el valor del índice entre corchetes, por ejemplo: vector[0] o vector[4].

Es importante también tener en cuenta que el espacio de memoria obtenido para almacenar los valores de un *array* será contiguo, esto es, toda la memoria usada por un array tendrá direcciones consecutivas, y no estará fragmentada.

Otro detalle muy importante es que cuando se declaran *arrays*, los valores para el número de elementos deben ser siempre constantes enteras. Nunca se puede usar una variable para definir el tamaño de un *array*.

Se pueden usar tantas dimensiones (índices) como queramos, el límite lo impone sólo la cantidad de memoria disponible.

Cuando sólo se usa un índice se suele hablar de vectores, cuando se usan dos, de tablas.

Los *arrays* de tres o más dimensiones no suelen tener nombres propios.

Ahora podemos ver que las cadenas de caracteres son un tipo especial de *arrays*. Se trata en realidad de *arrays* de una dimensión de objetos de tipo **char**.

Los índices son números enteros, y pueden tomar valores desde 0 hasta <número de elementos>-1. Esto es muy importante, y hay que tener mucho cuidado, porque no se comprueba si los índices son válidos.

Por ejemplo:

```
int Vector[10];
```

Creará un *array* con 10 enteros a los que accederemos como *Vector*[0] a *Vector*[9].

Como índice podremos usar cualquier expresión entera.

C++ no verifica el ámbito de los índices. Para poder hacerlo, el compilador tendría que agregar código, ya que los índices pueden ser variables, su valor debe ser verificado durante la ejecución, no durante la compilación. Esto está en contra de la filosofía de C++ de crear programas compactos y rápidos. Así que es tarea nuestra asegurarnos de que los índices están dentro de los márgenes correctos.

Si declaramos un *array* de 10 elementos, no obtendremos errores al acceder al elemento 11, las operaciones de lectura no son demasiado peligrosas, al menos en la mayoría de los casos. Sin embargo, si asignamos valores a elementos fuera del ámbito declarado, estaremos accediendo a zonas de memoria que pueden pertenecer a otras variables o incluso al código ejecutable de nuestro programa, con consecuencias generalmente desastrosas.

Ejemplo:

```
int Tabla[10][10];
char DimensionN[4][15][6][8][11];
...
DimensionN[3][11][0][4][6] = DimensionN[0][12][5][3][1];
Tabla[0][0] += Tabla[9][9];
```

Cada elemento de *Tabla*, desde *Tabla*[0][0] hasta *Tabla*[9][9] es un entero. Del mismo modo, cada elemento de *DimensionN* es un carácter.

Inicialización de arrays

Los *arrays* pueden ser inicializados en la declaración.

Ejemplos:

```
float R[10] = {2, 32, 4.6, 2, 1, 0.5, 3, 8, 0, 12};
float S[] = {2, 32, 4.6, 2, 1, 0.5, 3, 8, 0, 12};
int N[] = {1, 2, 3, 6};
int M[][3] = { 213, 32, 32, 32, 43, 32, 3, 43, 21};
char Mensaje[] = "Error de lectura";
char Saludo[] = {'H', 'o', 'l', 'a', 0};
```

Cuando se inicializan los *arrays* en la declaración no es obligatorio especificar el tamaño para la primera dimensión, como ocurre en los ejemplos de las líneas 2, 3, 4, 5 y 6. En estos casos la dimensión que queda indefinida se calcula a partir del número de elementos en la lista de valores iniciales.

El compilador sabe contar y puede calcular el tamaño necesario de la dimensión para contener el número de elementos especificados.

En el caso 2, el número de elementos es 10, ya que hay diez valores en la lista.

En el caso 3, será 4.

En el caso 4, será 3, ya que hay 9 valores, y la segunda dimensión es 3: 9/3=3.

Y en el caso 5, el número de elementos es 17, 16 caracteres más el cero de fin de cadena.

Operadores con arrays

Ya hemos visto que se puede usar el operador de asignación con *arrays* para asignar valores iniciales.

El otro operador que tiene sentido con los *arrays* es **sizeof**.

Aplicado a un *array*, el operador **sizeof** devuelve el tamaño de todo el array en bytes. Podemos obtener el número de elementos, si lo necesitamos, dividiendo ese valor entre el tamaño de uno de los elementos.

```
int main()
{
  int array[231];
  int nElementos;

  nElementos = sizeof(array)/sizeof(int);
  nElementos = sizeof(array)/sizeof(array[0]);
  return 0;
}
```

Las dos formas son válidas, pero la segunda es, tal vez, más general.

La utilidad de esta técnica es, como mucho, limitada. Desde el momento en que se deben usar constantes al declarar los *arrays*, su tamaño es siempre conocido, y por lo tanto, su cálculo es predecible.

Algoritmos de ordenación, método de la burbuja

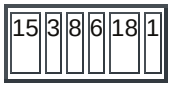
Una operación que se hace muy a menudo con los *arrays*, sobre todo con los de una dimensión, es ordenar sus elementos.

Uno de los más usados, aunque no de los más eficaces. Se trata del método de la burbuja.

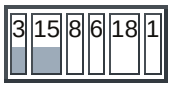
Este método consiste en recorrer la lista de valores a ordenar y compararlos dos a dos. Si los elementos están bien ordenados, pasamos al siguiente par, si no lo están los intercambiamos, y pasamos al siguiente, hasta llegar al final de la lista. El proceso completo se repite hasta que la lista está ordenada.

Lo veremos mejor con un ejemplo:

Ordenar la siguiente lista de menor a mayor:

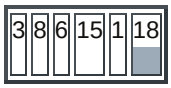


Empezamos comparando 15 y 3. Como están mal ordenados los intercambiamos, la lista quedará:

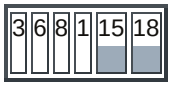


Tomamos el siguiente par de valores: 15 y 8, y volvemos a intercambiarlos, y seguimos el proceso...

Cuando lleguemos al final la lista estará así:



Empezamos la segunda pasada, pero ahora no es necesario recorrer toda la lista. Si observas verás que el último elemento está bien ordenado, siempre será el mayor, por lo tanto no será necesario incluirlo en la segunda pasada. Después de la segunda pasada la lista quedará:



Ahora es el 15 el que ocupa su posición final, la penúltima, por lo tanto no será necesario que entre en las comparaciones para la siguiente pasada. Las sucesivas pasadas dejarán la lista así:

EJERCICIOS PRáCTICOS

Problemas (creo que ya podemos empezar :-)

1. Hacer un programa que lea diez valores enteros en un *array* desde el teclado y calcule y muestre: la suma, el valor promedio, el mayor y el menor.

2. Hacer un programa que lea diez valores enteros en un *array* y los muestre en pantalla. Después que los ordene de menor a mayor y los vuelva a mostrar. Y finalmente que los ordene de mayor a menor y los muestre por tercera vez. Para ordenar la lista usar una función que implemente el método de la burbuja y que tenga como parámetro de entrada el tipo de ordenación, de mayor a menor o de menor a mayor. Para el *array* usar una variable global.

3. Hacer un programa que lea 25 valores enteros en una tabla de 5 por 5, y que después muestre la tabla y las sumas de cada fila y de cada columna. Procura que la salida sea clara, no te limites a los números obtenidos.

4. Hacer un programa que contenga una función con el prototipo bool Incrementa(char numero[10]);. La función debe incrementar el número pasado como parámetro en una cadena de caracteres de 9 dígitos. Si la cadena no contiene un número, debe devolver **false**, en caso contrario debe devolver **true**, y la cadena debe contener el número incrementado.

Si el número es "999999999", debe devolver "0". Cadenas con números de menos de 9 dígitos pueden contener ceros iniciales o no, por ejemplo, la función debe ser capaz de incrementar tanto la cadena "3423", como "00002323".La función *main* llamará a la función *Incrementar* con diferentes cadenas.

5. Hacer un programa que contenga una función con el prototipo bool Palindromo(char palabra[40]);. La función debe devolver **true** si la palabra es un palíndromo, y **false** si no lo es.

Una palabra es un palíndromo si cuando se lee desde el final al principio es igual que leyendo desde el principio, por ejemplo: "Otto", o con varias palabras "Anita lava la tina", "Dábale arroz a la zorra el abad". En estos casos debemos ignorar los acentos y los espacios, pero no es necesario que tu función haga eso, bastará con probar cadenas como "anitalavalatina", o "dabalearrozalazorraelabad".

La función no debe hacer distinciones entre mayúsculas y minúsculas.