Tipos estructurados de datos

Consultar el Capítulo 7 del libro: Programación en C++. Algoritmos, estructuras de datos y objetos, Joyanes, L., pag 275. Disponible en la sección de bibliografía de nuestra aula virtual.

Los arreglos existen en la inmensa mayoría de los lenguajes que ustedes van aprender en la universidad y en el resto de su vida profesional. Así que al estudiar este tema en C/C++, nuestra ganancia es doble.

Arreglos en C/C++

Muchos problemas son muy difíciles o imposibles de resolver si se utilizan solamente variables que pueden almacenar un único valor discreto, de cualquier tipo que sea. Por ejemplo, ¿qué haríamos si deseamos realizar ciertas tareas con un conjunto de datos, como listarlos, clasificarlos, promediarlos, buscar valores que cumplan características específicas, etc.? Pues, obviamente, lo primero que hay que hacer es reservar el espacio de memoria para contener cada uno de esos datos.

Nos haríamos entonces las siguientes preguntas:

¿Cuantos datos van a ser?, ya que tenemos que declarar esa cantidad de variables.

Supongamos que nos dicen que van a ser 15 datos. Entonces declararíamos las variables *a*, *b*, ..., *o*, para contener los quince números, y escribiríamos las operaciones ajustadas para realizar los cálculos con quince datos. Por ejemplo:

prom = (a + b + c + ... + o)/15;

Este programa nos podría funcionar para la primera corrida, pero ¿qué pasa si luego nos piden correr el programa para promediar 30 datos?, ¿y si en la tercera corrida nos piden promediar solo ocho datos?

La respuesta parecería sencilla: habría que ajustar las lecturas de datos y las fórmulas y luego compilar antes de correr el programa nuevamente. Pero esto es muy tedioso y es una muestra de que nuestro programa no está adaptado a cualquier situación.

En cuanto al manejo de conjuntos de datos, sin importar su tamaño, existen una serie de recursos, uno de los cuales es el uso de lo que se conoce como **Arreglos**. También recibe el nombre de **Vectores** (aunque no tiene nada que ver con los vectores de la matemática). Podemos explicarnos el arreglo como un tipo especial de variable que, en lugar de almacenar solo un dato, puede almacenar muchos datos al mismo tiempo.

El arreglo es una estructura de datos que permita manejar muchos datos al mismo tiempo.

Definición:
Un arreglo unidimensional se define como una colección finita, homogénea y ordenada de elementos. *Finita*, pues todo arreglo tiene un límite, es decir, se debe determinar el número máximo de elementos que forman parte del mismo. *Homogénea*, pues todos los elementos del arreglo son del mismo tipo. *Ordenada*, pues se puede determinar qué posición ocupa cada elemento del arreglo.

Por ejemplo, si tenemos el arreglo *A*, con la posibilidad de almacenar *n* valores, podríamos verlo almacenado en memoria como se muestra a continuación:

A:

6	15	4	. . .	21	8
0	1	2			n-1

Notar que esta variable contiene una serie de casillas (celdas de memoria) contiguas, en cada una de las cuales se puede almacenar un valor. Este es un arreglo unidimensional, lo podemos visualizar de manera horizontal, o de manera vertical, según nos convenga. Eso es indiferente. También podemos definir arreglos bidimensionales, en cuyo caso se les llama también tablas o matrices; tridimensionales; o de mayor dimensión. La cantidad de dimensiones no tiene límite, todo depende de la necesidad que se tenga para resolver un problema.

Repito nuevamente, en un arreglo se pueden almacenar varios valores simultáneamente porque contiene, no una, sino una serie de casillas de memoria. La cantidad de casillas que contendrá un arreglo se define en el momento de la declaración del mismo.

En cualquier arreglo podemos distinguir dos partes: las casillas y los subíndices. Para acceder a algún elemento del arreglo debemos utilizar:

a) El nombre del arreglo, el cual se ha asignado en el momento de la declaración.

b) El subíndice de la posición que ocupa el elemento que nos interesa. Este subíndice es un número entero que se le asigna a cada casilla de manera automática, comenzando desde cero. Hace referencia a la posición de cada dato almacenado en el arreglo.

La declaración de un arreglo en C/C++ es así:

Tipo Nombre[*Longitud dim 1*][*Longitud dim 2*]…[*Longitud dim n*];

Por ejemplo, el arreglo A, de arriba, se debió haber declarado así:

int A[Número entero];

Para hacer referencia a un dato específico del arreglo *A*, debe enunciarse el nombre del arreglo seguido de un par de corchetes y un subíndice en medio de ambos corchetes, así:

A[0] hace referencia al elemento en la casilla con subíndice 0, que es el primer elemento del arreglo;
A[1] hace referencia al elemento en la casilla con subíndice 1, que es el segundo elemento del arreglo;
A[2] hace referencia al elemento en la casilla con subíndice 2, que es el tercer elemento del arreglo.
Etc.

Obviamente, utilizaremos arreglos cuando queramos manejar conjuntos de datos que están relacionados entre sí. Por ejemplo, un arreglo para almacenar de edades, o un arreglo para almacenar salarios, o un arreglo para temperaturas, etc. El arreglo permitirá que todos los datos estén accesibles en memoria al mismo tiempo y facilitará que a todos los elementos se les pueda dar el mismo tratamiento. Por ejemplo, si se quiere calcular el promedio de los elementos de un arreglo, éste deberá recorrerse para ir sumando todos los valores.

Entonces, para procesar todos los elementos de un arreglo necesitamos de una instrucción iterativa. Notaremos que los arreglos y las instrucciones *for* se llevan muy bien, ya que el mismo subíndice del lazo nos sirve como subíndice del arreglo. Esto lo veremos en unos ejemplos a continuación.

Antes de pasar a los ejemplos debe decirse que los programas que utilizan arreglos deben estar preparados para recibir cualquier cantidad de elementos, de acuerdo al tipo de problema que se espera resolver. Es decir, en la declaración del arreglo se coloca un valor que sea suficientemente alto como para que sea muy difícil sobrepasar la cantidad de casillas necesaria. Por ejemplo, si vamos a hacer un programa para procesar las edades de estudiantes de las aulas de la UCA, y sabemos que el aula más grande tiene capacidad de 300 estudiantes, bien podríamos declarar un arreglo de 325 o 350 casillas para que aún quede un tanto sobrado de la cantidad máxima posible. Obviamente, si se quiere construir un programa robusto hay que poner validaciones con instrucciones *if* por si el usuario quiere sobrepasar la cantidad máxima de casillas disponibles.

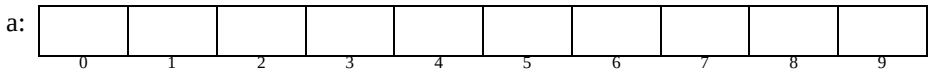
Otra forma de resolver esto es solicitar la cantidad de elementos con la que se va a trabajar, capturar ese valor en una variable entera y luego declarar el arreglo colocando esa variable dentro de los corchetes. Así el arreglo se va a definir del tamaño exacto para esa corrida del programa. Ojo, algunos compiladores podrían no permitir esta forma de declarar arreglos.

Ejemplos:

1) Declarar un arreglo de enteros, que pueda contener un máximo de diez elementos:

```
int a[10];
```

El arreglo de este ejemplo se visualiza de la siguiente manera:

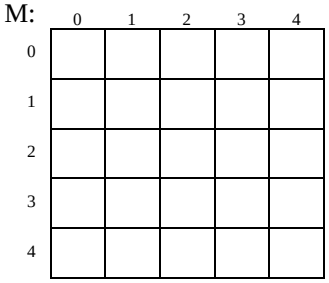


Notar que contiene diez casillas, tal y como dice la declaración, pero están numeradas de 0 a 9.

2) Para declarar un arreglo de dos dimensiones en el que podamos almacenar una tabla, o matriz, de datos de 5 x 5:

```
int M[5][5];
```

Este arreglo se visualiza de la siguiente manera:



3) En el siguiente ejemplo se declara un arreglo de enteros y al mismo tiempo se inicializa con valores. Luego se recorre el arreglo para imprimir su contenido.

```
#include <iostream>
using namespace std;

int main(void)
{
     int i, a[ ] = {50, 22, 37, 86, 28, 30, 42};

     for(i = 0; i <= 6; i++)
          cout << a[i] << " ";

      cout << endl;
     return 0;
}
```

Notar que en la declaración no es necesario indicar la longitud pues el compilador adapta el tamaño del arreglo según la cantidad de elementos con que se está inicializando. Aunque se puede hacer si se desea, pero en este caso debe coincidir el número entre corchetes con la cantidad de elementos entre las llaves.

La verificación de la finalización del lazo puede escribirse así: **i <= 6**, o así: **i < 7**. Ambas tienen el mismo efecto.

4) Escriba un programa que llene un arreglo con *n* elementos y luego calcule la suma de los elementos almacenados en el arreglo. Utilice dos lazos independientes, primero lea los elementos y luego sume todos los elementos.

```
#include <iostream>
using namespace std;

int main(void)
{
     int i, a[8], suma;

     cout << endl;
     cout << "SUMAR ELEMENTOS DE UN ARREGLO" << endl << endl;

     cout << "Digite ocho enteros: ";
     for(i = 0; i < 8; i++)
          cin >> a[i];

     suma = 0;
     for(i = 0; i < 8; i++)
          suma = suma + a[i];
     cout << "La suma de los elementos de arreglo es: " << suma << endl;

     cout << endl;
     return 0;
}
```

5) Escriba un programa que llene un arreglo con n enteros que representan las edades de un grupo de personas y luego calcule el promedio de edad.

```
#include <iostream>
using namespace std;
```

```
int main(void)
{
    int i, edades[50], suma, prom, n;

    cout << endl;
    cout << "PROMEDIO DE EDADES" << endl << endl;

    cout << "¿Cuantas edades va a promediar: ";
    cin >> n;
    if (n > 50)
        cout << "No puede sobrepasar el arreglo. Programa finalizado." << endl;
    else{
        cout << "Digite las " << n << " edades:" << endl;
        for(i = 0; i < n; i++)
            cin >> edades[i];

        suma = 0;
        for(i = 0; i < n; i++)
            suma = suma + edades[i];
        cout << "La suma de los elementos de arreglo es: " << suma << endl;
        prom = suma / n;
        Cout << "El promedio de edad es: " << prom << endl;
    }

    cout << endl;
    return 0;
}
```

Notar que el arreglo se ha declarado para contener un máximo de 50 valores. Así que en las diferentes ejecuciones del programa se pueden promediar 50 edades o menos. Si se promedian menos de 50 edades, habrá casillas sobrantes, que tendrán valores indeterminados que proporcionarán datos erróneos. Por ejemplo, si se promedian 35 edades, solo las primeras 35 casillas serán usadas, y las últimas 15 casillas no deberán accederse. El control del subíndice y el valor máximo que debe alcanzar, también es algo que se debe controlar.
En resumen, hay dos cosas que debemos poder controlar cuando trabajamos con arreglos:

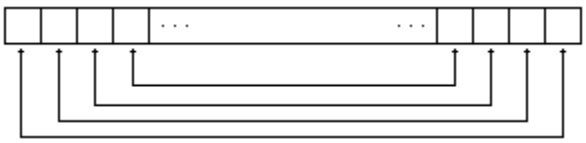
- Que no sobrepasemos la cantidad máxima de casillas del arreglo. Esto lo controlamos en este ejemplo en la condición de la instrucción if: if (n > 50)...
- Que no accedamos a las casillas que no se utilizarán. Esto lo controlamos en los límite del for: for(i = 0; i < n; i++).

Ejercicios:

1) Escriba un programa que genere los primeros n términos de la serie de Fibonacci y los almacene en un arreglo. Esta serie inicia con 0 y 1. Para determinar el k-ésimo elemento, se procede así:

$$a_k = a_{k-1} + a_{k-2}$$

2) Escriba un programa que lea **n** números, los guarde en un arreglo y luego invierta el orden de los elementos en el arreglo.



3) Escriba un programa que llene dos arreglos con valores provenientes desde teclado. Luego utilice un tercer arreglo para colocar, en cada casilla de este, la suma de los elementos de las casillas con mismo subíndice de los dos arreglos leídos.

De igual forma podemos trabajar con un arreglo bidimensional. En este caso, en lugar de manejar una hilera de valores, estaremos manejando una especie de tabla de valores, conocida también como matriz o determinante. A continuación se mencionan algunas cosas importantes que debemos tener claras al manejar arreglos unidimensionales y bidimensionales.

Si para recorrer un arreglo lineal necesitamos un lazo, entonces para recorrer un arreglo bidimensional, necesitaremos dos lazos **anidados**, cada uno con su propio subíndice.

```
for(i = 0; i < nFil: i++)
    for(j = 0; j < nCol; j++)
    ...
```

En un arreglo lineal en el que no utilizamos todas sus casillas, llenamos solo las primeras, a partir del subíndice cero. Las demás nos quedan sobrantes. El siguiente esquema nos muestra un arreglo lineal de 15 casilla en el que solo se utilizan las primeras 5 casilla:

3	7	-1	4	-6										

En un arreglo bidimensional, en el que solo usamos una parte de él, estaremos utilizando las casillas de la esquina superior izquierda. Este ejemplo nos muestra un arreglo bidimensional que se ha declarado de 10x10, pero que solo se está utilizando una parte para manejar una tabla de 3x3:

8	7	4							
0	4	-2							
-1	5	3							

Las demás casillas son espacio de memoria que el programa ha reservado en el momento de declarar esta variable, pero no se utilizará en esta corrida de programa concreta.

Finalmente, una matriz también se puede inicializar en el momento de su declaración, un ejemplo sería el siguiente:

```
int a[ ][3] = {{2, 7, -5}, {21, 8, 0}, {13, 46, 4}};
```

La longitud de la primera dimensión puede omitirse porque C/C++ la determina de acuerdo a la misma inicialización que se está realizando. Pero las longitudes de las demás dimensiones deben ser especificadas. Si se colocan las longitudes de sus dimensiones, deben coincidir con la cantidad de valores que se indica en la inicialización. En este caso se crea una matriz de 3x3, que contendrá esos valores que se indican desde su creación.

Paso de arreglos como argumentos a funciones

A diferencia de la variables simples, que ya hemos trabajado a lo largo de todo el ciclo, el nombre de una variable tipo arreglo es un puntero a la dirección de inicio del arreglo. Por ejemplo, en la declaración:

```
int a;
```

El identificador a es el nombre del espacio de memoria donde se guardarán números enteros.

Pero si declaramos el arreglo:

```
int b[25];
```

El identificador b, no solo es el nombre del arreglo, sino que es un puntero al primer elemento de la lista de veinte cinco enteros. Así que b contiene la dirección de la primera celda de memoria RAM que contiene al primero de los enteros del arreglo.

Lo anterior es importante conocerlo porque cuando queramos enviar como argumento un arreglo a una función, no necesitamos precederlo del operador "dirección de", &. Otra consecuencia de esto es que los arreglos siempre se envían por referencia. Jamás se puede enviar un arreglo por valor a una función, así que debemos de tener cuidado en no modificar el arreglo en la función que lo recibe, si no es necesario.

Sea el arreglo b declarado arriba, para enviarlo como argumento a una función, realizaremos la llamada así:

```
funcion(b, ...)
```

La cabecera de la función es la siguiente:

```
tipo funcion(int x[ ], ...)
```

Notar que, en la cabecera de la función el parámetro del arreglo no tiene la longitud especificada dentro de los corchetes.

Si el arreglo es de dos o más dimensiones no es necesario especificar la longitud de la primera dimensión en la cabecera de la función. Por ejemplo:

Sea el arreglo bidimensional:

```
int a[4][4];
```

La cabecera de la función que lo recibe como parámetro es la siguiente:

```
tipo func(int x[ ][4], ...)
```

Dentro la la función *main*, la llamada a esta función será:

```
func(a, ...)
```

Cuando pasamos un arreglo como argumento a una función, también podemos hacer que el parámetro que lo recibe sea un puntero del mismo tipo del arreglo. Este puntero recibirá la dirección de inicio del arreglo. Luego, en el cuerpo de la función, podremos desplazar el puntero a lo largo del arreglo con algo que se conoce como "aritmética de punteros". Esto lo veremos más adelante.