ARRAYS

Introducción

Un array o arreglo es un conjunto finito de elementos del mismo tipo de dato, almacenados en posiciones contiguas de memoria. Es una forma de declarar muchas variables del mismo tipo que pueden ser referenciadas por un nombre común y subíndices haciendo que el acceso a cada una de las variables pueda generalizarse mediante estructuras repetitivas.

El tipo de dato de los elementos se denomina tipo base del arreglo. Los arrays en el lenguaje C pueden ser de una o más dimensiones.

Ejemplos:

```
int a[10]; //define un array de una dimensión (vector) que puede almacenar 10 variables enteras

int b[5][10]; // define un array de dos dimensiones (matriz) que puede almacenar en total 50 elementos.
```

Los array en el lenguaje C en realidad guardan una dirección de memoria. Esa dirección de memoria corresponde a la dirección del primer elemento del conjunto de datos definido. Luego, mediante los subíndices se calcula la dirección del elemento puntual al que se quiere acceder. En el lenguaje C, las variables que guardan direcciones de memorias se denominan punteros, porque al contener una dirección de memoria pueden “apuntar” a otra variable, es decir guardar la dirección, la referencia de donde se encuentra otra variable. El tema de punteros está fuera del alcance de esta materia. Solo debe saber entonces que el identificador del vector guarda la dirección de memoria de inicio del mismo ya que es un puntero denominado puntero estático, debido a que esa dirección no se puede cambiar una vez definida. Esto dará un comportamiento particular cuando se envíe un array como parámetro de función.

Vectores

Los vectores son arrays de una dimensión en el cual cada elemento se identifica con el nombre del conjunto y un subíndice que determina su ubicación relativa en el mismo.

Para definir un vector se utiliza la siguiente notación:

```
tipo_de_dato_base  nombre_del_vector [cantidad de elementos];
```

Donde **cantidad de elementos** define la capacidad del vector, es decir, la cantidad máxima de elementos que puede almacenar.

En la declaración:

```
int ve [100]
```

`ve` es el nombre de un vector de 100 elementos cuyo tipo base es int (o sea, un vector de 100 enteros).

El acceso a cada elemento del vector se formaliza utilizando el nombre genérico (nombre_del_vector) seguido del subíndice encerrado entre corchetes. Por ejemplo, ve[i] hace referencia al elemento i del vector. Luego, la información almacenada puede recuperarse tanto en forma secuencial (asignando valores al subíndice desde 0 hasta la capacidad -1) como en forma directa con un valor arbitrario del subíndice que esté dentro de dicho rango.

El subíndice define el desplazamiento del elemento con respecto al inicio del vector; puede ser una constante, una variable, una expresión o, inclusive, una función en la medida que resulte ser un entero comprendido entre 0 y la capacidad del vector menos 1, dado que representa el desplazamiento del elemento con respecto al inicio del vector.

Ej.: VECTOR[5], VECTOR[M], VECTOR[M – K + 1], siempre que 5, M y M – K + 1, respectivamente, sean un entero entre 0 (cero) y cantidad de elementos - 1.

Podemos representar gráficamente un vector int v[12] de la siguiente manera:

v	23	12	5	-57	0	45	1	-96	32	7	10	30

v [0] : 23; v[4] : 0; v[7] : -96; El último elemento es v[11]: 30

Arreglos como parámetros de funciones

Como se mencionó anteriormente al definir una variable del tipo array (ya sea un vector o una matriz), esa variable guarda la dirección de memoria de inicio del conjunto de datos. Cuando se envíe como parámetro de una función entonces un array lo que se envía y se copia en una variable local a la función es esa misma dirección, es decir, que dentro de la función se sigue referenciando a la misma dirección de memoria que está reservada para el conjunto de datos en el bloque desde el cual se invoca a la función. Esta particularidad hace que los cambios que se hagan sobre el array dentro de la función se van a ver reflejados desde donde se llamó a la función ya que trabaja directamente sobre la memoria del mismo.

Para definir un vector como parámetro formal de una función se indica el tipo base, el nombre y luego un par de corchetes que es lo que identifica al parámetro como un arreglo. Ejemplos:

```
int ve[], char vc[], float vf[]
```

No es necesario especificar el tamaño del vector ya que solo se copia la dirección de memoria de inicio y NO los datos del vector.

Vectores Paralelos

Muchas veces para resolver un problema es necesario guardar más de un dato de una misma entidad y por lo tanto con un solo vector no será suficiente. Por ejemplo si se desea ingresar la lista de precios de un negocio donde por cada producto se tenga el código y su precio será necesario definir dos vectores uno que guarde el código del producto y un segundo vector que guarde los precios. Estos vectores estará relacionados es decir que en la posición 0 del vector de códigos se guardará el código del producto y en la posición 0 del vector de precios se guardará el precio de ese mismo producto. Es decir que los vectores están relacionados guardando en los elementos con igual subíndice información de una misma entidad. A estos vectores se los denomina vectores paralelos o apareados.

En el siguiente ejemplo se muestra vectores paralelos donde se cargó la información de productos.

`int vcod[4]` `float vprecios[4]`

vcod
1500
5878
2565
1566

vprecios
10.55
17.35
5.40
20.30

Búsqueda en vectores

Frecuentemente es necesario buscar un determinado elemento en un vector, ya sea para determinar si el mismo está en el vector o para recuperar la posición en donde se encuentra.

Los datos del problema son:

- El vector.
- La cantidad de elementos que tiene.
- El elemento a buscar.

Las salidas son:

- Un indicador de la existencia del elemento en el arreglo
- La posición donde se encuentra.

Podríamos unificar ambas salidas, si existe, la función devuelve la posición donde se encuentra (0 ≤ posición < cantidad de elementos) y si no existe retornará un -1 (valor inválido como posición).

Búsqueda Secuencial

Consiste en ir recorriendo uno a uno los elementos del vector hasta que encontremos el buscado.

```
int secuencial(int v[], int t, int bus)
{
   int i=0;
   while(i<t && v[i]!=bus)
      i++;
   if(i==t)
      return -1;
   else
      return i;
}
```

Búsqueda Binaria

La búsqueda binaria funciona en arreglos ordenados. La búsqueda binaria comienza por comparar el elemento del medio del arreglo con el valor buscado. Si el valor buscado es igual al elemento del medio, su posición en el arreglo es retornada. Si el valor buscado es menor o mayor que el elemento del medio, la búsqueda continua en la primera o segunda mitad, respectivamente, dejando la otra mitad fuera de consideración.

```
int binaria(int v[], int t, int bus)
{
   int pos = -1, desde = 0,hasta = t-1, medio;
   while(desde <= hasta  &&  pos == -1)
   {
      medio = (desde+hasta)/2;
      if(v[medio] == bus)
         pos = medio;
      else
      {
         if(bus < v[medio])
            hasta = medio-1;
         else
            desde = medio+1;
      }
   }
   return pos;
}
```

Ordenamiento de vectores

Los métodos de ordenamiento son numerosos. Podemos considerar dos grandes grupos:

- Directos: Burbujeo
  Selección
  Inserción
- Indirectos (avanzados): Shell
  Ordenación por mezcla (Merge Sort)
  Ordenamiento Rápido (Quicksort)

En listas pequeñas, los métodos directos se comportan en forma eficiente, mientras que en vectores con gran cantidad de elementos, se debe recurrir a métodos avanzados.

Método de ordenamiento por **burbujeo o intercambio**

Supongamos que tenemos que ordenar en forma ascendente (de menor a mayor) un vector "a", mediante una función, conociendo sus elementos enteros y su dimensión. Una posible solución, sería comparar a[0] con a[1], intercambiándolos si están desordenados, lo mismo con a[1] y a[2], y así sucesivamente hasta llegar al último elemento.

Si nuestro vector está formado por los siguientes 5 elementos:

7	**26**	**8**	**15**	**9**
0	1	2	3	4

Al comparar a[0] con a[1] no se producen cambios: 7 < 26

7	**26**	8	15	9

Al comparar a[1] con a[2] : intercambio el 26 con el 8

7	**8**	**26**	15	9

Al comparar a[2] con a[3] : intercambio el 26 con el 15

7	8	**15**	**26**	9

Al comparar a[3] con a[4] intercambio el 26 con el 9, resultando:

7	8	15	**9**	**26**

De esta forma, al finalizar la primera pasada, en la última posición del arreglo, ha quedado el elemento mayor. El resto de los elementos mayores, tienden a moverse, "burbujean" hacia la derecha. Se trata ahora de comenzar nuevamente con las comparaciones de los elementos (segunda pasada) sin necesidad de tratar el último, ubicado en a[4].

Al comparar a[0] con a[1] y a[1] con a[2] no se producen intercambios.

7	8	15	9	26

Al comparar a[2] con a[3] intercambio el 15 con el 9, resultando:

7	8	**9**	**15**	26

Al finalizar la segunda pasada, con una comparación menos, ya obtuvimos en la anteúltima posición del arreglo el mayor entre los restantes (el valor 15).

Así sucesivamente, en cada ciclo la cantidad de comparaciones disminuye en 1.

De esta forma en la tercera pasada se comparará al elemento a[0] con a[1] y al a[1] con a[2]. En nuestro ejemplo no se producirán cambios:

7	8	9	15	26

En la cuarta y última pasada, dado que nuestro vector "a" contiene 5 elementos, tampoco se efectuaran cambios al comparar a[0] con a[1], asegurándonos que nuestro vector, ya resultó ordenado:

7	**8**	**9**	**15**	**26**

En resumen, para ordenar un vector de n elementos, son necesarias realizar como máximas n-1 pasadas, y en cada oportunidad una comparación menos.

La siguiente función resuelve según el método descripto

```
void burbujeo(int v[], int n)
{
   int i=1, j, aux;
   bool cambio;
   do
   {
      cambio=false;
      for(j=0; j  <  n-i; j++)
      {
         if(v[j] > v[j+1])
         {
            aux=v[j];
            v[j]=v[j+1];
            v[j+1]=aux;
            cambio=true;
         }
      }
      i++;
   }while(i < n && cambio);
}
```

Método de ordenamiento por **selección**

Su funcionamiento es el siguiente:

- Buscar el mínimo elemento de la lista
- Intercambiarlo con el primero
- Buscar el siguiente mínimo en el resto de la lista
- Intercambiarlo con el segundo

Y en general:

- Buscar el mínimo elemento entre una posición i y el final de la lista
- Intercambiar el mínimo con el elemento de la posición i

```
void seleccion(int v[],unsigned t)
{
   unsigned i,j;
   int menor, pos;
   for(i=0;i<t-1;i++)
   {
      menor=v[i];
      pos=i;
      for(j=i+1;j<t;j++)
      {
         if(v[j]<menor)
         {
            menor=v[j];
            pos=j;
         }
      }
      v[pos]=v[i];
      v[i]=menor;
   }
}
```

Método de ordenamiento por **inseción**

El **ordenamiento por inserción** es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria. Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay ***i*** elementos ordenados de menor a mayor, se toma el elemento ***i+1*** y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se *inserta* el elemento ***i+1*** debiendo desplazarse los demás elementos.

```
void insercion(int v[], int n)
{
   int i, a, j;
   for(i=1; i < n; i++)
   {
      a=v[i];
      j=i-1;
      while(j > = 0 &&  v[j] > a)
      {
         v[j+1]=v[j];
         j--;
      }
      v[j+1]=a;
   }
}
```

Matrices (arreglos bidimensionales)

Recordemos la definición de arreglos: Un arreglo es un conjunto finito de elementos del mismo tipo de datos, almacenados en posiciones contiguas de memoria. Bien, un arreglo bidimensional es un arreglo en el cual cada elemento es a su vez otro arreglo. Un arreglo llamado B, el cual consiste de M elementos, cada uno de los cuales es un arreglo de N elementos de un cierto tipo T se puede representar como una tabla de M filas y N columnas.

	0	1	.	.	j	.	N-1
0							
1							
.							
i					X		
.							
M-1							

Para identificar a cada elemento es necesario utilizar dos subíndices: uno para identificar la fila y otro para la columna. El primer subíndice hace referencia a la fila y el segundo a la columna. Así, el elemento marcado con X en la figura se identifica como B [i] [j]

A este tipo de estructura se lo llama ***matriz***