Vectores en C

Sintaxis

Definición

Un vector es un grupo de posiciones en memoria relacionadas entre sí porque tienen el mismo nombre y son del mismo tipo.

Para referirse a una posición en particular o elemento dentro del vector, especificaremos el nombre del vector y el número de posición del elemento particular dentro del mismo.

a: nombre del vector

0 1 2 3 4 5 6 posición

a [1] = 5 **contenido**

a es un vector de enteros que contiene 7 elementos, cualquiera de éstos elementos puede ser referenciado dándole el nombre seguido por el número de posición de dicho elemento en particular entre [ ].

El primer elemento de cualquier vector es el elemento de posición cero.

a [0], a [1],.., a [6]

El elemento de orden i se conoce como a [i-1].

El número que está entre los [ ] se llama subíndice y debe ser un entero o expresión entera.

Para multiplicar el tercer elemento por 4 y asignarlo a una variable x sería:

x = a [2] * 4;

¿Cómo declarar los vectores en C?

Se debe especificar el nombre del vector, el tipo y la cantidad de elementos.

```
int a [7];
```

guarda 7 lugares para guardar enteros o sea cada uno de 2 bytes.

Ejemplo:

Inicializar un vector de 5 elementos en cero y mostrarlos con formato tabulado:

```
# include < stdio . h >
int main ( )
{
int n [ 5 ];
int i;
for ( i = 0; i <5 ; i++ ) /*inicializa en cero*/
        n [ i ] = 0;
printf( " %8s%13s\n", "elemento","valor");
for ( i = 0; i<5; i++)
    printf ( "%8d%13d\n", i , n [ i ];
}
```

Los elementos de un vector también se pueden inicializar en la declaración del vector:

```
int a[6] = { 1, 3, 5, 3, 4, 6};
```

int a[6] = {0} inicializa el primer elemento a cero y los 5 restantes les pone en cero automáticamente.

Si en la declaración no se coloca el tamaño asume la cantidad de elementos que luego se cargan entre { }.

```
int vec[ ] = {1, 2, 8, 4};   asume  vec[4]
```

Operaciones con vectores:

1) Cargar
2) Recorrer
3) Mostrar
4) Buscar un elemento
5) Desplazar
6) Intercambiar
7) Ordenar

1) Cargar un vector de n1 elementos, siendo n1 a lo sumo 50.

Para cargar un vector se usa una instrucción repetitiva, por ejemplo for.

Si no se sabe exactamente la cantidad de elementos que tendrá el vector, lo declaramos con la cantidad máxima de elementos posibles.

```
int vec[50];
```

La cantidad de elementos que tendrá el vector la obtenemos ingresándola por teclado. Luego validamos que no supere la cantidad máxima.

```
do {
    printf ( "Ingrese la cantidad real de elementos");
    scanf ( "%d", &n1);
    } while (n1 > 50 || n1 < 1);
```

```
#include <stdio.h>
int main ( )
{
int vec [ 50 ];
int n1,i;
do {
    printf ( "Ingrese la cantidad real de elementos");
    scanf ( "%d", &n1);
     } while ( n1 > 50 || n1 < 1 );
for ( i=0 ; i < n1 ; i++ )
{
    printf ( " Ingrese la componente % d ", i ) ;
    scanf( " %d ", &vec [ i ] ) ;
}
}
```

2) Recorrer el vector

Utilizamos, en general, también un ciclo repetitivo for

Por ejemplo si queremos sumar los elementos del vector, el código es:

```
. . .
for ( i=0; i< n1; i++)
    sum  + = vec [ i ];
```

Por supuesto antes definimos sum como int y en cero.

3) Mostrar un vector

Para imprimirlo también usamos un for pues debemos recorrerlo para mostrarlo.

```
. . .
for ( i= 0; i < n1 ; i ++ )
        printf ( " vec [ %d ] = %d\n" , i , vec [ i ] ) ;
```

O usando tabulaciones:

```
. . .
printf ( " %8s% 13s \ n " , " elemento ", " valor ");
for ( i = 0 ; i < n1 ; i ++ )
        printf ( " %8d %13d\n" , i , vec [ i ] ) ;
. . .
```

4) Buscar un elemento en un vector

Como es posible que el elemento lo encontremos antes de terminar de recorrer el vector no es conveniente hacerlo con un for. Es preferible utilizar un while pues nos permite comparar con la condición y cortar la búsqueda al encontrarlo.

La búsqueda de un elemento no significa que ese elemento esté en el vector. Por eso la condición del while debe ser doble, una por la condición pedida y la otra para ver si la posición no llegó al último elemento.

Por ejemplo, para decir en qué posición está el elemento cuyo contenido es 10.

```
. . .
n=0;
while ( vec [ n ] ! = 10 && n < n1)
          n ++;
if (n == n1)
          printf ( " El valor 10 no se encuentra" ) ;
else
         printf ( " El valor 10 está en la %d posición" , n ) ;
```

Una vez que sale del while doble, debemos preguntar por cuál de las dos condiciones salió y accionar según corresponda.

5) Desplazar los elementos de un vector.

Para desplazar los elementos de un vector podemos hacerlo hacia delante (a la izquierda) o hacia atrás (a la derecha).

Si lo hacemos hacia delante un lugar, se pierde el primer valor ya que en la posición 0 queda lo que está en la posición 1 y así sucesivamente hasta la posición última que

quiero desplazar. Pero el valor que está en el último lugar queda repetido en la posición anterior, es por esto que muchas veces se aclara que hacer en esa posición, por ejemplo reemplazar por 0.

```
. . .
for ( i = 1; i < = posfinal ; i ++ )
    vec [ i-1 ] = vec [ i ];
// si me pide que reemplace por 0 colocaría :
    vec [ posfinal ] = 0;
. . .
```

El for comienza en 1 porque se asigna a la posición i - 1 y si i valiese 0 quedaría posición -1 que NO EXISTE. Si el desplazamiento es hasta posfinal se copia el valor a la posición anterior.

Si es más de un lugar el desplazamiento solo varía en lo que le restamos a la i y en el valor inicial del for.

Si lo hacemos hacia atrás un lugar, se pierde el último valor ya que en la posición posfinal queda lo que está en la posición anterior a esa y así sucesivamente hasta la posición primera que quiero desplazar. Pero el valor que está en el primer lugar queda repetido en la posición siguiente, es por esto que muchas veces se aclara que hacer en esa posición, por ejemplo reemplazar por 0.

En este caso tenemos que tener cuidado con el for, en que sentido? Si el for incrementa la variable, es decir si vamos asignando desde la primera posición a la última pedida, perderíamos todos los valores y lograríamos repetir el primer valor a lo largo de todo el intervalo. MAL!!!!

Que debemos hacer?

Recorrer el for de atrás hacia adelante

```
. . .
for ( i = posfinal -1; i > = posinicial; i -- )
        vec [ i+1 ] = vec [ i ];
// si me pide que reemplace por 0 colocaría :
        vec [ posinicial ] = 0;
. . .
```

El for comienza en posfinal-1 porque se asigna a la posición i+1 y si i valiese posfinal quedaría en una posición más que el máximo del vector que NO EXISTE.

6) Intercambiar

Para intercambiar dos elementos de un vector deben conocerse las dos posiciones que se quieren intercambiar.

Supongamos que quiero intercambiar la pos1 con la pos2.

Si ponemos

```
vec [ pos2 ] = vec [ pos1 ];
```

¡En la pos2 queda el valor que está en la pos1, PERO CUIDADO!!! El valor que tenía originalmente la pos2 se PERDIO y en la pos1 sigue quedando su valor original. Es decir no logramos lo pedido!!!

Para poder intercambiar, debemos declarar una variable auxiliar, para guardar allí el valor de pos2 antes de hacer la asignación.

Esto quedaría así:

```
aux = vec [ pos2 ];
vec [ pos2 ] = vec [ pos1 ];
vec [ pos1 ] = aux;
```

7) Ordenar:

Para ordenar un vector sobre sí mismo hay varios métodos uno de ellos llamado burbujeo es el siguiente:

Ejemplo:

Ordenar un vector de n1 elementos en forma ascendente.

```
. . .
int i, j, aux;
for ( i = 0; i < n1 -1; i++ )
        for ( j = i + 1; j < n1 ; j ++ )
                if ( vec [ i ] > vec [ j ] )
                {
                        aux = vec [ i ];
                        vec [ i ] = vec [ j ];
                        vec [ j ] = aux;
                }
. . .
```

¿Cómo pasar vectores en funciones?

Para pasar un vector como argumento real en la llamada de una función hay que especificar el nombre del vector sin los corchetes.

Ejemplo:

si se declara:

```
int vec [ 10 ];
```

La llamada a una función sería:

x = maximo (vec, n1);

Nombre del vector

Cantidad de elementos reales

pasa el vector y el tamaño a la función.

C pasa en forma automática los vectores a las funciones utilizando simulación de llamadas por referencia (las funciones llamadas pueden modificar los valores de los elementos en los vectores originales de los llamadores).

El nombre del vector es la dirección del primer elemento.

Si no deseo pasar el vector completo, sino un elemento del mismo lo debo pasar por valor, es decir, colocando el nombre y el subíndice o posición del elemento a pasar.

Por ejemplo:

pepe (vec [3]);

Ahora bien:

En el prototipo y en el desarrollo si un argumento es un vector se debe escribir con su tipo, nombre y los corchetes.

Para que una función reciba un vector a través de una llamada de función la lista de parámetros de la función debe especificar que se va a recibir un vector.

Por ejemplo:

el encabezado de la función máximo sería:

int maximo (int vec [ ], int n1)

indicando que máximo espera recibir un vector de enteros en vec y su cantidad de elementos en el parámetro entero n1.

No es necesario colocar el tamaño del vector entre los [ ].

El prototipo podría escribirse:

```
int máximo (int [ ], int);
```

Recordar: el prototipo le indica al C la cantidad de parámetros y su tipo

Problema: vectores con funciones

Dados 2 vectores de n1 elementos, siendo n1 de a lo sumo 50, calcular:

a) El vector C como la suma del vector a + vector b, es decir
b) c [i] = a [i] + b [i]
c) Indicar la posición del máximo elemento de a.
d) Reemplazar por 1 (uno) los elementos de posición par de b.
e) Ordenar el vector a en forma ascendente.

```
#include  <stdio. h>
void cargar ( int x [ ] , int n );
void mostrar ( int x [ ] , int n );
void vectorc  ( int a [ ] , int b [ ], int c [ ], int n1 );
int posmax  ( int a [ ] , int n1 );
void unos  ( int b [ ], int n1 );
void ordenar  ( int a [ ] , int n1 );
int main (  ){
int n1, a [50], b [50], c [50];
int i , x ;
do {
     printf ( "Ingrese la cantidad real de elementos");
     scanf ( "%d", &n1);
      } while ( n1 > 50 || n1 < 1 );
cargar ( a, n1 ) ;
printf ("VECTOR A\n") ;
mostrar ( a, n1) ;
cargar ( b, n1 ) ;
printf ("VECTOR B\n") ;
mostrar ( b, n1) ;
vectorc ( a, b, c, n1 );
x = posmax ( a, n1 )
printf ("\n En la posición %d del vector a está el máximo valor", x );
unos (b, n1 );
ordenar ( a, n1 );
printf ("VECTOR A ORDENADO\n");
mostrar ( a, n1 );
getch( );
}
```

```
void cargar ( int x [ ], int n )
{
int i;
for ( i = 0 ; i < n ; i ++ )
{
     printf ("\n Ingrese el elemento  %d del vector" , i );
     scanf ("%d", & x [ i ] );
}
return;
}

void  mostrar ( int  x [ ], int  n )
{
int i;
printf ("ELEMENTO    VALOR");
for ( i = 0 ; i < n ; i ++ )
          printf ( "% 8d %8d" , i , x [ i ] );
return;
}

void  vectorc ( int a [ ], int b [ ], int c [ ], int  n1 )
{
int i;
for ( i = 0 ; i < n1 ; i + + )
{
          c [ i ] = a [ i ] + b [ i ] ;
          printf("\n c [%d]= %d", i, c [ i ]);
}return;}
```

```
void unos  ( int b [ ], int n1 )
{
int i;
for ( i = 0; i < n1; i +=2)
        b [ i ] = 1;
for ( i = 0; i < n1; i++)
        printf("b[%d]= %d\n", i, b[i]);
return;
}

void ordenar(int a[ ], int n1)
{
int i, j, aux;
for ( i = 0 ; i < n1-1 ; i++)
        for ( j = i+1 ; j < n1; j++)
                if ( a [ i ] > a [ j ])
                    {
                        aux = a[ i ];
                            a[ i ] = a [ j ];
                                a [ j ]= aux;
                    }
return;
}
```