Universidad Nacional de la Matanza

Programación Inicial

UNIDAD 7. *INTRODUCCION A VECTORES*

INDICE

UNIDAD 7
Introducción a Vectores

OBJETIVOS: Tener un primer acercamiento a la utilización de vectores para almacenar y procesar datos.

1. Introducción

Un array es un conjunto finito de elementos del mismo tipo de dato, almacenados en posiciones consecutivas 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. Dentro de esta materia veremos los arrays de una dimensión llamados vectores.

El tamaño de los arrays debe ser definido al escribir el programa, es decir, que se debe conocer o estimar la cantidad máxima de elementos que va a necesitar almacenar para así dar un tamaño adecuado a los arrays.

En el lenguaje C los arrays se definen de forma similar a una variable poniendo el tipo y el identificador, pero se agrega luego del mismo, entre corchetes, la cantidad de elementos que contendrá el array en cada una de sus dimensiones. Por ejemplo:

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

Las variables del tipo 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 y 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 como se verá más adelante.

2. 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** es una constante que 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

Habitualmente, no se conoce exactamente cuántos elementos han de procesarse en un determinado programa y, además, en distintas ejecuciones de dicho programa, pueden procesarse diferentes volúmenes. Por ejemplo, si se define un vector para almacenar datos sobre los alumnos de un curso es evidente que distintos cursos tienen distinta cantidad de alumnos; en esos casos, se define un vector con capacidad suficiente para almacenar la información del curso más numeroso. Este dato suele no ser preciso por lo que generalmente se sobredimensiona el array (conviene no exagerar en el sobredimensionamiento pues esto significa mantener memoria ociosa). Así, si en una universidad se manejan cursos de 50 ó 60 alumnos, podría definirse el array de 100 elementos. Aún así, es necesario controlar, cuando se agregan elementos, que no se supere esa capacidad: Si quisiéramos agregar el elemento 101, éste ocupará memoria que no está reservada para el conjunto con lo cual pueden destruirse otras variables del programa.

> ***El lenguaje C no chequea los límites de los arreglos. Por lo tanto, es responsabilidad del programador cuidar que el programa se mantenga siempre dentro de los límites definidos.***

Debe mantenerse siempre una variable donde guardar la cantidad de elementos que realmente tiene almacenado el vector.

int v[10] ; int cant_elem;

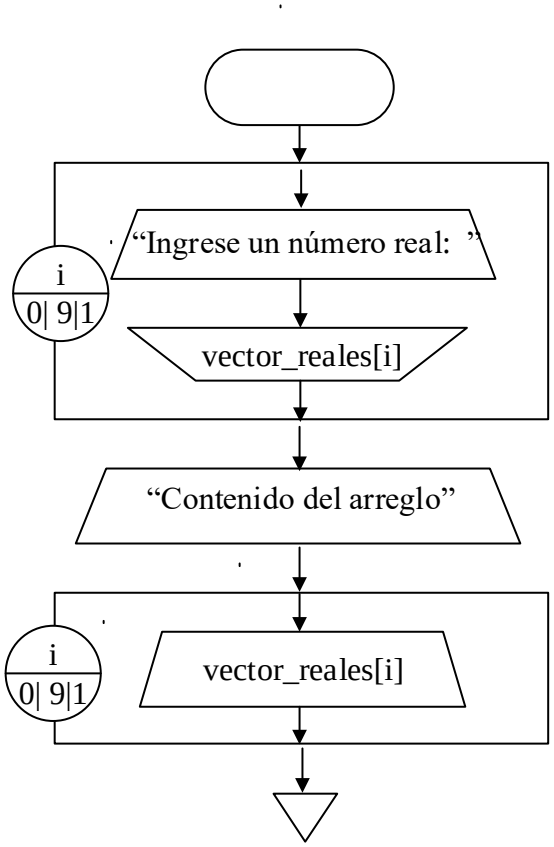
v:	23	1	0	38	15	227	4			
	0	1	2	3	4	5	6	7	8	9

cant_elem | 7 |

Note que en las posiciones 7, 8 y 9, si bien en el gráfico no se especificó contenido alguno por ser éste irrelevante, el mismo existe: siempre hay algo en cualquier dirección de memoria y, obviamente, por ser un vector de enteros, lo que hay se interpretará como un entero. Esto significa que si se recorre el arreglo desde 0 hasta cant_elem -1, siempre se encontraran valores enteros

aunque no se hayan colocado en el transcurso del programa, esos datos no los tenemos que tener en cuenta. Decimos de forma genérica que en esas posiciones hay “basura” ya que son datos no controlados.

Ejemplo 1: Generar un vector de 10 números reales leyendo dichos valores del teclado y luego mostrarlo:



Codificación en C

```
#include <stdio.h>
int main()
{
    float vector_reales[10];
    int i;
   // Lee los 10 números reales y los ubica secuencialmente en el arreglo
    for(i=0; i<=9; i++)
    {
        printf("\nIngrese un numero real: ");
        scanf("%f", &vector_reales[i]);
    }
    // Muestra el contenido del arreglo
    printf("\n Contenido del arreglo\n");
    for(i=0; i<=9; i++)
    {
        printf("%6.2f\t", vector_reales[i]);
    }
}
```

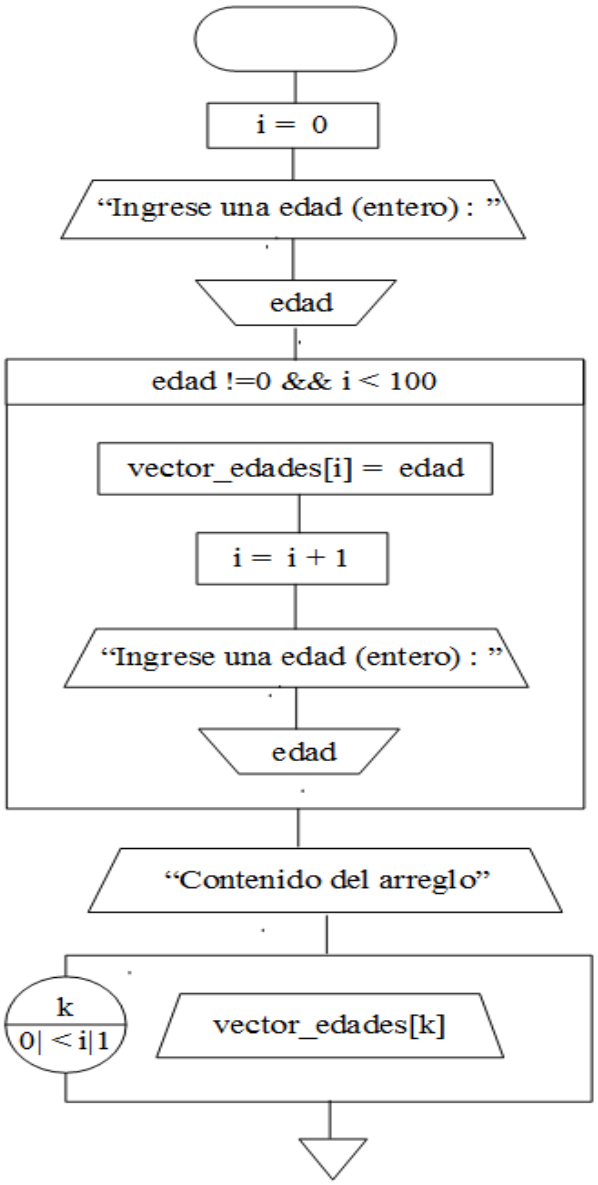
La salida de este programa es:

```
Ingrese un numero real: 23
Ingrese un numero real: 31
Ingrese un numero real: 3.25
Ingrese un numero real: .98
```

```
Ingrese un numero real: 5.5
Ingrese un numero real: 678
Ingrese un numero real: 500.60
Ingrese un numero real: 12
Ingrese un numero real: 0
Ingrese un numero real: 1

Contenido del arreglo
23.0  31.00    3.25    0.98    5.50  678.00  500.60   12.00    0.00    1.00
```

Ejemplo 2: Generar un vector con edades de los alumnos de un curso que se ingresan por teclado. Se estima que los cursos no tienen más de 100 alumnos. La carga finaliza con una edad igual a 0.

En este caso, no se conoce cuántos datos se van a leer; se asume que, como máximo, hay 100. Se define entonces un vector de 100 enteros pero debe verificarse en la carga de la información que no se supere dicho valor.



Nota importante: si bien puede ingresarse un elemento directamente en el arreglo, en este caso no debe procederse así pues, si se ingresaran más de 100 elementos, el elemento 101 se colocaría en la posición 100 del vector, que es memoria no reservada para el mismo. Luego, se ingresa el

dato en una variable auxiliar y, una vez confirmada la disponibilidad de espacio, se lo pone en el arreglo.

Codificación en C

```
#include <stdio.h>

int main()
{
	int vector_edades[100];
	int edad, i, k;
	i=0;
	printf("\nIngrese una edad (entero): ");
	scanf("%d", &edad);
	while (edad != 0 && i< 100)
	{
		vector_edades[i]=edad;
		i++;
		printf("\nIngrese una edad (entero): ");
		scanf("%d", &edad);
	};

	/* queda el arreglo armado con, a lo sumo,100 edades.
	   La cantidad real quedo en i */

	printf("\n Contenido del arreglo\n");
	for(k=0; k<i; k++)
	{
		printf("%5d", vector_edades[k]);
	}
	return 0;
}
```

3. Inicialización de Vectores

Al momento de declarar una variable del tipo vector es posible asignarle valores, para ello se pueden detallar cada uno de sus elementos. Por ejemplo, la instrucción:

```
int vec[5] = {2,52,100,58,18};
```

Define un vector de 5 posiciones donde en cada lugar ya tiene un número asignado, quedando el vector de la siguiente manera

Vec	2	52	100	58	18

El mismo resultado se puede obtener con la siguiente instrucción:

```
int vec[] = {2,52,100,58,18};
```

Dejar vacío el tamaño del vector al declarar memoria SOLO es válido si se detallan todos sus elementos ya que de forma automática le asignará el tamaño según la cantidad de elementos especificados en este caso será un vector de 5 posiciones.

Un caso muy habitual es utilizar cada posición de un vector como un contador o un acumulador, en dicho caso y para no poner muchos ceros como datos iniciales se puede escribir:

```
int vCont[10] = {0};
```

Esta instrucción crea un vector de 10 posiciones y pone todas las posiciones del mismo en 0. De esta forma rápidamente se puede poner en 0 un vector de cualquier tamaño.

Pero supongamos ahora que necesitamos que cada posición del vector se va a utilizar para calcular una productoria (es decir el contenido se lo multiplica por otro valor y se lo vuelve a guardar). En dicho caso será necesario que todas las posiciones del vector comiencen en 1 y no en 0 ya que si lo inicializamos en 0 al multiplicarlo por si mismo siempre daría 0. Si escribimos esta instrucción:

```
int vProd[10] = {1};
```

Obtendríamos el siguiente resultado:

1	0	0	0	0	0	0	0	0	0

Es decir que pone el 1 en la primera posición, pero el resto como no está especificadas la completa con 0. Entonces para inicializar un vector de 10 posiciones todo con 1 tendríamos que escribir:

```
int vProd[10] = {1,1,1,1,1,1,1,1,1,1};
```

Esto puede traer confusiones y más cuando son vectores aún más grandes. Lo recomendable en estos casos es recorrer el vector posición a posición y asignar el número 1 en cada una de ellas. El siguiente programa declara un vector de 10 elementos y lo inicializa todo con el valor 1.

```
int main()
{
      int vProd[10],i;
      for (i=0;i<10;i++)
            v[i] = 1;
}
```

Es importante recordar que la inicialización directa del vector (usando las llaves) SOLO puede utilizarse al definir la variable, por lo tanto si en el medio del programa un vector tiene que ser puesto por ejemplo nuevamente en 0, se debe utilizar sí o sí el método de recorrerlo y asignarle a cada posición el dato deseado.

Si se define un vector de mayor cantidad de los datos que se inicializan, el resto de los datos se completan con 0. Este ejemplo:

```
int vec[10] = {1,2,1,8};
```

genera el siguiente vector:

1	2	1	8	0	0	0	0	0	0

4. 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. Entonces cuando se envíe un array como parámetro de una función, 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

Por ejemplo si en el programa principal se define un vector de 5 elementos y se desea invocar una función para que solicite los datos por teclados y los guarde en el vector, a la función se puede enviar el vector y la cantidad de datos a completar. Una posible cabecera para dicha función sería: void CargarVector (int v[], int N). Donde V es la dirección de inicio del vector donde se van a guardar los datos y N es la cantidad de elementos a solicitar. Nótese que la función no retorna ningún valor (void) ya que los datos cargados en el vector dentro de la función se verán reflejados en el programa principal porque se trabaja directamente sobre la memoria de este. La Figura 1 muestra el esquema de memoria durante el pasaje de parámetros (las direcciones de memoria son ficticias ya que su largo depende de la arquitectura del procesador de la computadora donde se ejecute el programa. El dato 5 no sale de una variable de la memoria principal ya que se envía como una constante en la llamada a la función. Además, en la función se declara una variable local i que servirá de subíndice para recorrer el vector.

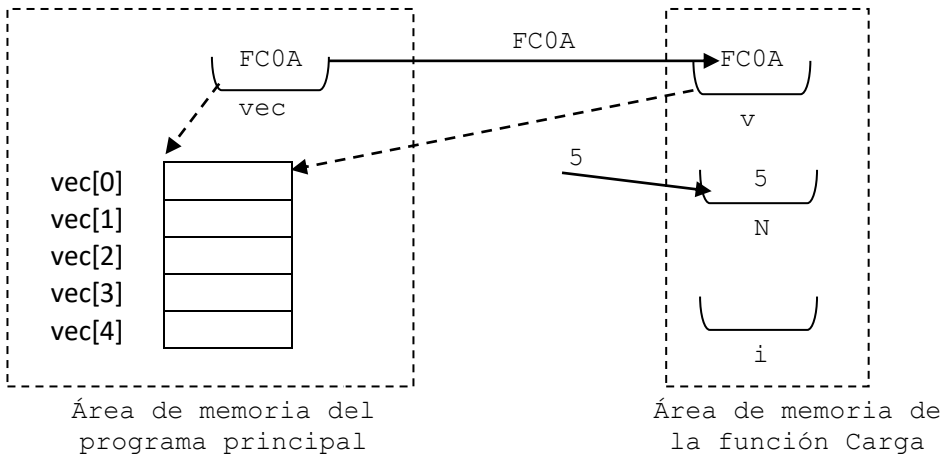
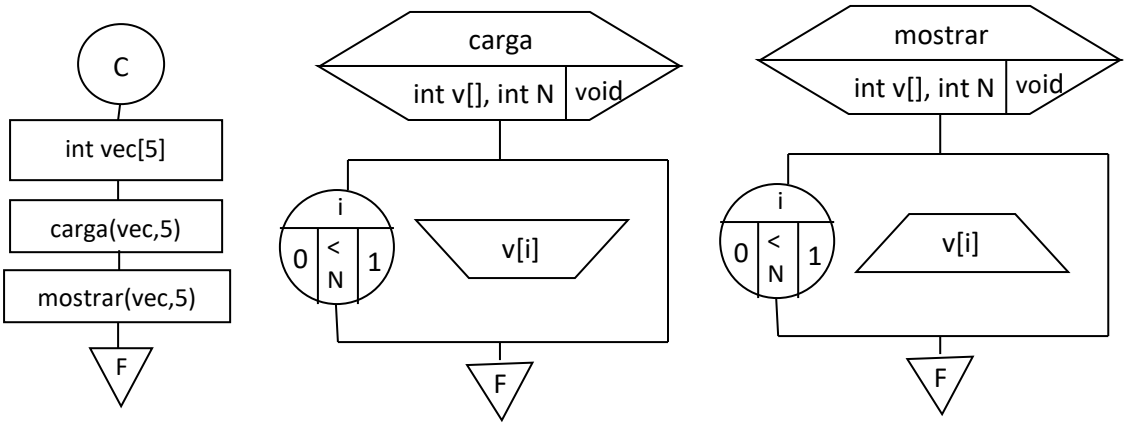


Figura 1: esquema de memoria al enviar un vector como parámetro a una función

Al enviar solo la dirección de memoria del vector, la misma función carga puede reutilizarse para cargar vectores de distinto tamaño siempre que sean del mismo tipo de dato. Esta función servirá igualmente para cargar un vector de 5, uno de 10 o uno de 1000 elementos por ejemplo, simplemente se debe declarar una variable con la capacidad suficiente en el programa principal y enviar en N la cantidad de elementos a cargar.

A continuación, se muestra un programa que, utilizando dos funciones, carga un vector de 5 elementos y los muestra por pantalla (si bien en el diagrama no es necesario declarar la variables a utilizar, es recomendable hacerlo en el caso de los arreglos para conocer la cantidad de datos disponibles).

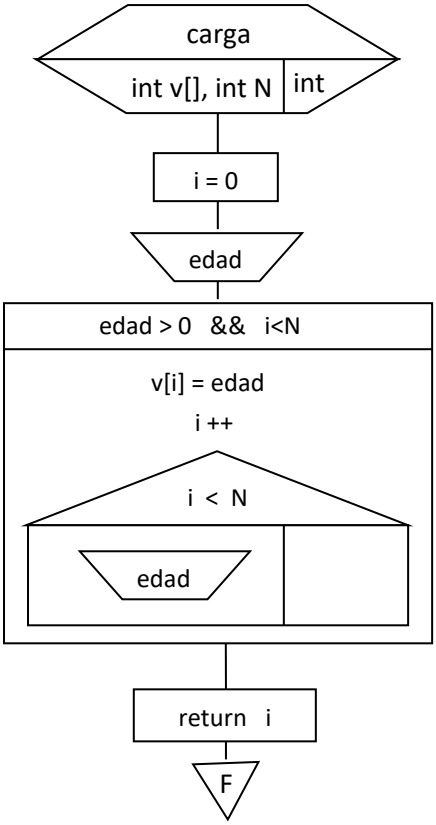


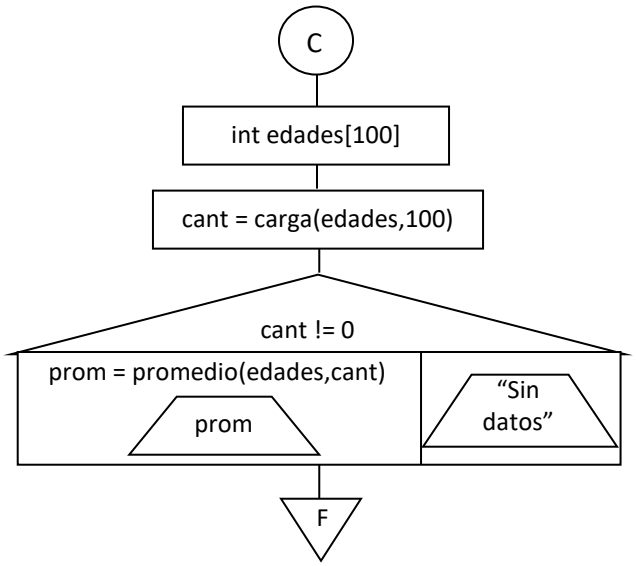
Codificación en C:

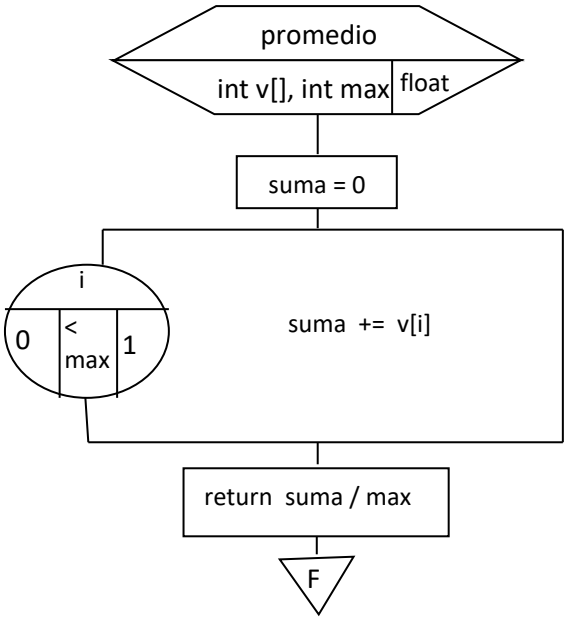
```
#include <stdio.h>
void carga(int[], int);
void mostrar(int[], int);
int main ()
{
    int vec[5];
    carga(vec,5);
    mostrar(vec,5);
    return 0;
}

void carga(int v[], int N)
{
    int i;
    for (i=0;i<N;i++)
    {
        printf("Ingrese un numero: ");
        scanf("%d",&v[i]);
    }
}

void mostrar(int v[], int N)
{
    int i;
    for (i=0;i<N;i++)
        printf("%d\n",v[i]);
}
```

Muchas veces no se sabe exactamente la cantidad de datos a ingresar por lo que se dimensiona el vector con un tamaño suficiente y se debe realizar una función de cargar que complete los datos sin pasarse del tamaño del vector, pero esta función debe además retornar la cantidad de elementos realmente ingresados.

A modo de ejemplo realizaremos el siguiente programa: ingresar las edades de los alumnos de un curso. Calcular luego el promedio de edades. No se sabe la cantidad exacta de alumnos, pero sí que no son más de 100. El ingreso de datos finaliza con una edad menor o igual a 0.







Como puede verse la función de carga retorna la cantidad de datos ingresada y luego este dato se pasa a la función que calcula el promedio para que no tome posiciones del vector que no tengan datos válidos. El promedio solo debe calcularse sí se ingresaron datos, caso contrario se produciría una división por cero, generando un error de ejecución en el programa.

Dentro del ciclo repetitivo de la función de carga se agrega una condición para no solicitar una edad sino queda espacio en el vector para guardarla.

Código C:

```
#include <stdio.h>
int carga(int[], int);
float promedio (int[], int);
int main()
{
    int edades[100], cant;
    float prom;
    cant=carga(edades,100);
    if (cant!=0)
    {
        prom = promedio(edades, cant);
        printf("El promedio de edades del curso es %.2f", prom);
    }
    else
        printf("Sin Datos");
    return 0;
}

int carga(int ve[], int N)
{
    int i=0,edad;
    printf ("Ingrese la edad del alumno:");
    scanf("%d",&edad);
    while (edad>0 && i<N)
    {
        ve[i]=edad;
        i++;
        if (i<N) //evita ingresar un dato cuando no hay espacio para almacenarlo
        {
            printf ("Ingrese la edad del alumno:");
            scanf("%d",&edad);
        }
    }
    return i;
}

float promedio (int v[], int max)
{
    int suma=0,i;
    for (i=0;i<max;i++)
        suma+=v[i];
    return (float)suma/max;
}
```

5. 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 estarán relacionados, ya 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.

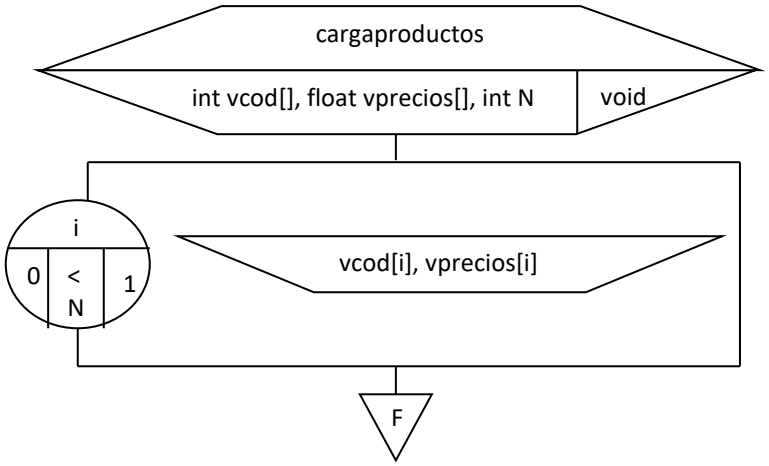
La Figura 2 muestra un ejemplo donde en vectores paralelos se cargó la información de los productos.

int vcod[4]	float vprecios[4]
1500	10.55
5878	17.35
2565	5.40
1566	20.30

Figura 2: vectores en paralelo con código y precio de productos

El producto con código 1500 tiene un precio de 10.55 mientras que el producto con código 5878 tiene un precio de 17.35 y así sucesivamente siempre la información está relacionada en ambos vectores.

Al momento de realizar una función de carga para este tipo de problema se debe solicitar la información completa de la entidad y guardarla en los vectores, es decir que se ingresa primero el código y precio del primer producto luego código y precio del segundo y NO primero todos los códigos y luego todos los precios ya que de esa forma sería confuso para el usuario. A continuación, se muestra una función de carga de la información de los productos donde se solicita el código y el precio de cada uno.




```
void cargaProductos(int vcod[], float vprecios[], int N)
{
    int i;
    for (i=0;i<N;i++)
    {
        printf("Ingrese el código de producto: ");
        scanf("%d",&vcod[i]);
        printf("Ingrese el precio del producto %d: ", vcod[i]);
        scanf("%f",&vprecios[i]);
    }
}
```