
# *Elementos de Programación*

# *UNIDAD 9. ESTRUCTURA DE DATOS*

## INDICE

# UNIDAD 9 - ESTRUCTURA DE DATOS

**OBJETIVOS:** *Poder definir nuevos tipos de datos compuestos, para guardar información de una misma entidad en una única variable. Reemplazar el uso de vectores paralelos con vectores de estructuras.*

## 1. Como crear una estructura de datos en el lenguaje C.

Cuando se inicia la construcción (codificación) de programas en los llamados programas de alto nivel se comienza con algoritmos cuyos datos están alojados en lo que se conoce como variables de "tipo simple": enteros, reales, caracteres, etc. (en Lenguaje C, int, float, char, etc.). Pero en ciertos casos se torna complicado el manejo de las variables en forma separada debiendo generar muchas variables para guardar la información de una entidad. Además, cuando se trabajan con varias entidades se deben generar varios vectores paralelos para guardar la información. Por ejemplo, si se tiene un producto que dispone de un código numérico, una descripción de texto y un precio, ya son 3 variables diferentes que guardan la información de la misma entidad, y se necesitan más variables si se quiere por ejemplo guarda el stock y otros datos de dicho producto. El lenguaje C permite crear una estructura de datos donde se puede guardar toda la información de determinada entidad en una única variable, definiendo un nuevo tipo de dato.

El tipo de dato que se pasará a desarrollar se lo suele llamar "estructura de datos" o "registro" (en el lenguaje C "struct") y es el agrupamiento de un conjunto de datos simples y muchas veces involucra el anidamiento de estructuras de datos "struct" como se verá más adelante.

La estructura de datos o registro se lo puede ver mejor a través de un ejemplo: Suponiendo que se quiere procesar los datos de una persona, y se establece tener la siguiente información:

| | |
|---|---|
| DNI | Como la variable que contiene el Documento Nacional de Identidad (tipo entero, en Lenguaje C es int) |
| ApellNombre | Como la variable que contiene el Apellido y el Nombre (suponiendo una cadena de 40 caracteres incluyendo 1 carácter más por el fin de cadena (\0). |
| Sexo | Como la variable que contiene el Sexo de la persona (tipo carácter, en Lenguaje C es char) donde F corresponde a Femenino y M corresponde a Masculino. |
| DiaNacimiento | Como la variable que contiene el Día de Nacimiento (tipo entero, en Lenguaje C es int) |
| MesNacimiento | Como la variable que contiene el Mes de Nacimiento (tipo entero, en Lenguaje C es int) |
| AnioNacimiento | Como la variable que contiene el Año de Nacimiento (tipo entero, en Lenguaje C es int) |

Los datos de una persona podrían ser muchos más, pero por una cuestión de simplificación y didáctica se utilizarán los enunciados en los párrafos anteriores para una estructura de datos.

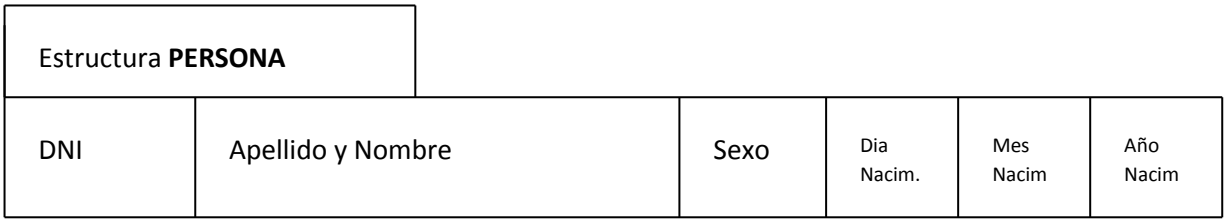
En el ejemplo, se pueden observar que son variables simples (variables: int, float, char). Si se codifican los siguientes datos en el Lenguaje C, sería de la siguiente manera (Ejemplo 1):

Ejemplo 1:

```
#include <stdio.h>
int main()
{
  //declaración de variables locales al programa principal
  int DNI;
  char ApellNombre[41], Sexo;
  int DiaNacimiento, MesNacimiento, AnioNacimiento;
  //posteriormente irán las propias sentencias del programa principal
  return 0;
}
```

En el "Ejemplo 1" el acceso a las distintas variables es en forma separada, pero las mismas no darán el concepto de unidad de datos que representan el registro de una persona.

En el gráfico se puede observar cómo se agrupan los mismos datos del "Ejemplo 1" de forma tal que formen una estructura de datos o registro.




En el Lenguaje C la forma para poder lograr el agrupamiento descrito arriba se lo conoce con el nombre de la palabra reservada "struct "y se codifica de la siguiente manera:

Ejemplo 2:

```
#include <stdio.h>

struct PERSONA
{
    int DNI;
    char ApellNombre[41];
    char Sexo;
    int DiaNacimiento;
    int MesNacimiento;
    int AnioNacimiento;
} ;
int main()
{
    …
}
```

Este es el nombre que le asigna al agrupamiento de datos que llamará PERSONA. Y se lo denomina ***"etiqueta de estructura"***

**Importante:** "No olvidar" en la llave de cierre de la estructura la colocación del ";" (punto y coma).

Cada uno de los datos dentro de la estructura se lo conoce como "campos" o "miembros" los mismos están englobados/encerrados entre llaves las cuales son obligatorias.

Cabe aclarar que la estructura de datos o registro se declarará dentro del área de declaraciones de variables globales por lo que tendrá una aplicación amplia en el programa principal y en todas las funciones del programa. También puede declararse dentro de funciones, pero esa estructura solo será válida dentro de dicha función ya que su declaración sería local.

**Nota**: Es importante destacar que la declaración de una estructura de datos o registro no es una variable que ocupa un lugar en memoria, pero es en si misma "un molde o un sello" que permitirá la declaración de variables que contengan esa estructura de datos o registro. Para declarar una o varias variables de esta estructura de datos ("struct PERSONA") se deberá proceder como se verá en el "Ejemplo 3", dentro del main o programa principal.

Ejemplo 3:

```
struct PERSONA
{
    long int DNI;
    char ApellNombre[41];
    char Sexo;
    int DiaNacimiento;
    int MesNacimiento;
    int AnioNacimiento;
} ;
int main()
{
// declaración de variables locales al programa principal

{
  struct PERSONA empleado;


  //posteriormente irán las propias sentencias del programa principal
  return 0;
}
```

Aquí se declara la variable **empleado** que ocupa lugar en memoria y que obedece a la estructura **PERSONA**.

**Nota**: No se pueden repetir los nombres de los miembros (campos) dentro de la declaración de una misma estructura de datos.

## 2. Como acceder a los miembros (campos) de una estructura de datos.

Para poder tener acceso a un determinado miembro (campo) de una estructura de datos se procede de la siguiente forma:

**nombre de la variable tipo estructura**. (punto) **nombre del miembro**

Ejemplo 4:

/* Es un programa que permite ingresar datos por teclado y guardarlos en una variable tipo estructura llamada PERSONA y luego mostrarlos por pantalla. Se podrá observar la utilización el operador miembro de estructura " . " ( punto) */

```
#include <stdio.h>
#include <conio.h>
struct  PERSONA   // Aqui se declara la estructura de datos
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   int DiaNacimiento;
   int MesNacimiento;
   int AnioNacimiento;
} ;
```

```
int main()
{
   /* Aquí se declara la variable "empleado" de tipo de la estructura de datos
   "struct PERSONA"*/
   struct PERSONA empleado;

   /* Aquí se van a ingresar por teclado el contenido de los diferentes campos que
   contiene la variable "empleado" */
   printf("Ingrese Numero de DNI");
   scanf("%d", &empleado.DNI);
   printf("Ingrese Numero de Apellido y nombre");
   fflush(stdin);
   gets(empleado.ApellNombre);
   printf("Ingrese el Sexo");
   fflush(stdin);
   scanf("%c", &empleado.Sexo);
   printf("Ingrese Numero del Dia de Nacimiento");
   scanf("%d", &empleado.DiaNacimiento);
   printf("Ingrese Numero del Mes de Nacimiento");
   scanf("%d", &empleado.MesNacimiento);
   printf("Ingrese Numero del Año de Nacimiento");
   scanf("%d",  &empleado.AnioNacimiento);
   printf("El empleado cuyo DNI es :  %d    , se llama  %-40s    y nacio el dia
   %2d/%2d/%4d",    empleado.DNI ,  empleado.ApellNombre,  empleado.DiaNacimiento,
   empleado.MesNacimiento , empleado.AnioNacimiento);
   return 0;
}
```

## 3. Como inicializar los miembros (campos) de una variable tipo estructura.

La inicialización de variables simples con un determinado valor se procede de la siguiente manera:

```
int acum = 0;
char caracter = 'F';
char cadena[21] = "Hola que Tal? ";
```

Para inicializar los miembros (campos) de una variable tipo estructuras, se procede de la siguiente manera:

```
struct PERSONA empleado = { 37234546, "Martinez, Julia", 'F',
18, 11, 1982};
```

**Nota**: Se debe respetar y conservar el orden y la totalidad de los miembros (campos) encerrados entre llaves { } (corchetes), y también sus correspondientes características, en el ejemplo utilizado serán:

```
(long int ,  cadena de caracteres ("    "), carácter (' ') ,
int, int , int);
```

**Nota**: Se debe respetar las capacidades máximas de cada uno tipo de dato de los miembros (campos) de una variable de estructura de dato.

```
#include <stdio.h>
#include <conio.h>
struct PERSONA // Aqui se declara la estructura de datos
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   int DiaNacimiento;
   int MesNacimiento;
   int AnioNacimiento;
};
```

```
int main()
{
   // Aquí se declara la variable "empleado" y se le asignan valores a
   cada uno de sus miembros (campos)
   struct PERSONA empleado = { 37234546, "Martinez, Julia", 'F', 18, 11,
   1982};

   printf("El empleado cuyo DNI es : %d, se llama %-40s y nacio el dia
   %2d/%2d/%4d",empleado.DNI,                    empleado.ApellNombre,
   empleado.DiaNacimiento,          empleado.MesNacimiento          ,
   empleado.AnioNacimiento);
   return 0;
}
```

## 4. Como utilizar Arreglos de estructuras de datos (vectores y matrices).

Se pueden generar los arreglos (vectores) y/o matrices con los elementos del tipo estructura de datos (**struct**).

La definición de estructuras hace que la información de una entidad esté unificada en una única variable, y como la definición de la estructura forma un nuevo tipo de dato también es posible definir vectores de estructuras eliminando la necesidad de utilizar vectores paralelos para guardar información relacionada, ahora cada posición del vector podrá contener todos los datos que sean necesarios.

En el "Ejemplo 5" se plantea un arreglo (vector) de 10 posiciones que contiene datos tipo de estructura de datos (struct).

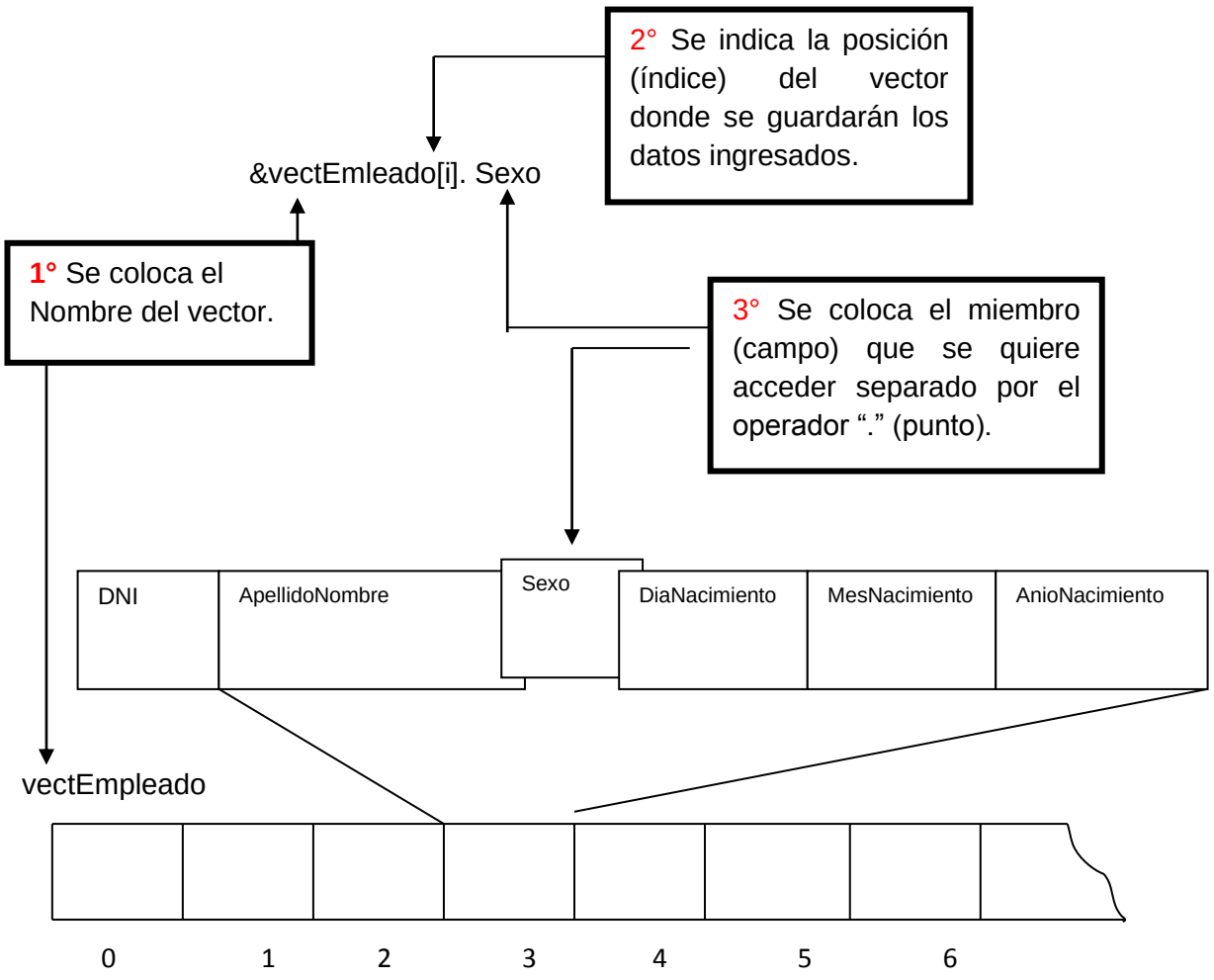
Ejemplo 5:

/* Es un programa que permite ingresar datos en el vector de 10 elementos del tipo de dato PERSONA */

```
#include <stdio.h>
#include <conio.h>
struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   int DiaNacimiento;
   int MesNacimiento;
   int AnioNacimiento;
};


int main()
{
   struct PERSONA vectEmpleado[10]; // Aquí se declara la variable vector de 10
   elementos tipo struct PERSONA
   int i;

   // Aqui se ingresan los datos por teclado y se guardan en el vector
   for (i= 0 ;  i < 10 ;  i++)
   {
      printf("Ingrese Numero de DNI");
      scanf("%d", &vectEmpleado[i].DNI);
      printf("Ingrese Numero de Apellido y nombre");
      fflush(stdin);
```

```
        gets(vectEmpleado[i].ApellNombre);
        printf("Ingrese Sexo");
        fflush(stdin);
        scanf("%c", &vectEmpleado[i].Sexo);
        printf("Ingrese Numero de Dia Mes Año de Nacimiento");
        printf("\nIngrese Numero del Dia de Nacimiento");
        scanf("%d", & vectEmpleado[i].DiaNacimiento);
        printf("Ingrese Numero del Mes de Nacimiento");
        scanf("%d", & vectEmpleado[i].MesNacimiento);
        printf("Ingrese Numero del Año de Nacimiento");
        scanf("%d",  & vectEmpleado[i].AnioNacimiento);
    }

    //  Aquí se muestra el contenido de los elementos del vector.
    for (i= 0 ;  i < 10 ;  i++)
    {
        printf("El empleado cuyo DNI es :  %d    , se llama  %-40s    y nacio el dia
        %2d/%2d/%4d\n"  ,    vectEmpleado[i].DNI  ,    vectEmpleado[i].ApellNombre  ,
        vectEmpleado[i].DiaNacimiento    ,       vectEmpleado[i].MesNacimiento       ,
        vectEmpleado[i].AnioNacimiento);
    }
    return 0;
}
```

A continuación, se podrá observar en forma gráfica la secuencia de anidamiento de los datos ingresados por teclado en un arreglo (vector) de tipo estructura de datos.



## 5. Como Anidar una de estructura de datos.

Las estructuras de datos permiten su anidamiento, con lo cual dentro de una estructura de datos puede haber una o varias estructuras de datos dentro de ella.

El “Ejemplo 5” podría modificarse para crear una estructura de datos que contenga la fecha de nacimiento, o sea generar una nueva estructura de datos que contenga el Dia, Mes y Año. Este proceso se puede visualizar en el “Ejemplo 6”.

Para visualizar y entender esta nueva ampliación se tomará en cuenta los miembros que conforman la fecha de nacimiento es decir el día, mes y año, y con estos datos se dará lugar a la formación de la típica estructura FECHA, y será una reutilización permanente porque se le podrá dar diferentes usos, por ejemplo: fecha de nacimiento, fecha de casamiento, fecha de ingreso, etc., todas ellas serán del mismo tipo de estructura.

Ejemplo 6:

/* Es un programa que permite ingresar por teclado los datos en la estructura PERSONA y a su vez mostrarlos por pantalla. Aquí se podrá ver cómo se utiliza el operador miembro de estructura “. “(punto) */

```
#include <stdio.h>
#include <conio.h>

// Aqui se declara la estructura de datos con el nombre FECHA
struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
// Aqui se declara la estructura de datos con el nombre PERSONA
struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   struct FECHA Nacimiento;
   /* Aqui se crea una variable con los campos de la estructura de datos con el
   nombre FECHA. Cabe aclarar que la estructura de datos FECHA tiene que estar
   declarada previamente, de lo contrario daría un error de compilación.*/
} ;
int main()
{
   struct PERSONA empleado;
   printf("Ingrese Numero de DNI");
   scanf("%d", &empleado.DNI);
   printf("Ingrese Numero de Apellido y nombre");
   fflush(stdin);
   gets(empleado.ApellNombre);
   printf("Ingrese Sexo");
   fflush(stdin);
   scanf("%c", &empleado.Sexo);
   printf("Ingrese Numero del Dia de Nacimiento");
   scanf("%d", &empleado.Nacimiento.Dia);
   printf("Ingrese Numero del Mes de Nacimiento");
   scanf("%d", &empleado.Nacimiento.Mes);
   printf("Ingrese Numero del Año de Nacimiento");
   scanf("%d", &empleado.Nacimiento.Anio);
   printf("El empleado cuyo DNI es :  %d    , se llama  %-40s    y nacio el dia
   %2d/%2d/%4d",    empleado.DNI ,  empleado.ApellNombre,  empleado.Nacimiento.Dia
   ,   empleado.Nacimiento.Mes , empleado.Nacimiento.Anio);
   return 0;
}
```

En el “Ejemplo 6” se realizó la codificación de un ejercicio donde se accede a un miembro (campo) por medio de estructuras de datos anidadas, utilizando para ello tantos operadores miembros (“.”, punto) como niveles de anidamiento haya.

Aquí se puede observar en forma gráfica como quedaría el anidamiento de estructuras:

Estructura **PERSONA**

<table>
<tr><td rowspan="2">DNI</td><td rowspan="2">Apellido y Nombre</td><td rowspan="2">Sexo</td><td colspan="3">Nacimiento</td></tr>
<tr><td>Dia</td><td>Mes</td><td>Año</td></tr>
<tr><td colspan="3"></td><td colspan="3">Estructura FECHA</td></tr>
</table>

## 6. Como utilizar una estructura de datos en funciones.

Al formar un nuevo tipo de dato una estructura de datos puede utilizarse como parámetros. En el Ejemplo 7, se recibe como parámetro una estructura de datos que contiene una fecha y la función validará si la misma corresponde a una fecha correcta devolviendo: un entero “1” si es incorrecta o un “0” si es correcta.

Ejemplo7:

1- Llamada en el programa principal (función main):

```
esFechaCorrecta (fecha);
```

2- Prototipo de la función:

```
int esFechaCorrecta(struct FECHA );
```

3- Declaración de la función:

```
/* Aquí se coloca el encabezado de la función en donde se encuentra como parámetro
de entrada una variable tipo estructura FECHA. */

int esFechaCorrecta (struct FECHA fecha)
{
   int retorno, bisiesto, cantidaddiasmes;
   retorno = 0;
   if(fecha.Anio%4==0 && fecha.Anio %100!=0|| fecha.Anio %400==0)
      bisiesto=1;
   else
      bisiesto =0;
   if(fecha.Mes==4|| fecha.Mes ==6|| fecha.Mes ==9|| fecha.Mes ==11)
      cantidaddiasmes =30;
   else
   {
      if(fecha.Mes ==2)
         cantidaddiasmes =28+ bisiesto;
      else
         cantidaddiasmes =31;
   }
   if(fecha.Anio   >=1900&&   fecha.Mes   >=1&&   fecha.Mes   <=12&&   fecha.Dia>=1&&
   fecha.Dia<= cantidaddiasmes )
      retorno =1;
   else
      retorno =0;
   return retorno;
}
```

En el “Ejemplo 8” se puede observar la construcción de una función que retorna una variable tipo estructura FECHA, que contendrá en cada uno de sus miembros (campos) los datos correspondientes a una fecha.

Ejemplo 8:

1- Llamada en el programa principal (función main ):

```
struct FECHA f;
f = IngresoDeFecha ( );
```

2- Prototipo de la función:

```
struct FECHA IngresoDeFecha ( );
```

3- Declaración de la función:

```
struct FECHA IngresoDeFecha ( )
{
   // Aquí se declara una variable local tipo estructura fecha
   struct FECHA fecha;
   printf("Ingrese Numero del Dia de una fecha");
   scanf("%d", &fecha.Dia);
   printf("Ingrese Numero del Mes de una fecha");
   scanf("%d",  &fecha.Mes);
   printf("Ingrese Numero del Año de una fecha");
   scanf("%d", &fecha.Anio);
   return fecha // Aquí se retorna la variable tipo estructura FECHA
}
```

En el “Ejemplo 9” se puede observar un programa en cual se utilizarán las funciones descriptas en el “Ejemplo 6” y “Ejemplo 7”.

Ejemplo 9:

Es un programa que permite ver el uso de estructuras en funciones.

```
#include <stdio.h>
#include <conio.h>
struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
struct FECHA IngresoDeFecha( );// Prototipo de la función
int esFechaCorrecta(struct FECHA );// Prototipo de la función
int main()
{
   struct FECHA fecha_main;// Declaración de una variable tipo estructura FECHA.
   int retorno;
   fecha_main = IngresoDeFecha();// llamada a la función
   retorno = esFechaCorrecta(fecha_main);
   if( retorno == 1)
      printf("La fecha es correcta");
   else
      printf("Es una fecha Incorrecta");
   return 0;
}

struct FECHA IngresoDeFecha( )
{
   struct FECHA fecha;
   printf("Ingrese Numero del Dia de una fecha:");
```

```
   scanf("%d", &fecha.Dia);
   printf("Ingrese Numero del Mes de una fecha:");
   scanf("%d",  &fecha.Mes);
   printf("Ingrese Numero del Anio de una fecha:");
   scanf("%d", &fecha.Anio);
   return fecha;// Aquí se retorna la variable tipo estructura FECHA
}

int esFechaCorrecta (struct FECHA fecha)
{
   int retorno, bisiesto, cantidaddiasmes;
   retorno = 0;
   if(fecha.Anio%4==0 && fecha.Anio %100!=0|| fecha.Anio %400==0)
      bisiesto=1;
   else
      bisiesto =0;
   if(fecha.Mes==4|| fecha.Mes ==6|| fecha.Mes ==9|| fecha.Mes ==11)
      cantidaddiasmes =30;
   else
   {
      if(fecha.Mes ==2)
         cantidaddiasmes =28+ bisiesto;
      else
         cantidaddiasmes =31;
   }
   if(fecha.Anio   >=1900&&   fecha.Mes   >=1&&   fecha.Mes   <=12&&   fecha.Dia>=1&&
   fecha.Dia<= cantidaddiasmes )
      retorno =1;
   else
      retorno =0;
   return retorno;
}
```

## 7. Copia de estructuras

Las estructuras forman un nuevo tipo de dato y por lo tanto permiten su manejo como una unidad. Si se tienen dos variables de la misma estructura es posible asignar directamente una a otra con el símbolo de asignación (=).

Automáticamente todos los campos de la estructura a la derecha del signo igual se copian a la segunda estructura sin importar de qué tipo sean dichos campos o si hay estructuras anidadas, vectores, etc. todos los campos se copian.

Ejemplo 10:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};

struct PERSONA
{
   long int DNI;
   char ApellNombre[41];
   char Sexo;
   struct FECHA Nacimiento;
} ;
```

```
int main()
{
  struct PERSONA p, copia;

  //asignación de datos a la variable p
  p.DNI = 23223122;
  strcpy(p. ApellNombre, "JUAN PEREZ");
  p.Sexo = 'M';
  p.Nacimiento.Dia = 10;
  p.Nacimiento.Mes = 5;
  p.Nacimiento.Anio = 1982;

  copia = p; //copia de la variable estructura

  printf("Los datos copiados son:\n Nombre: %s \n Sexo: %c \n DNI: %d \n Fecha de
nacimiento:    %d/%d/%d"    ,    copia.ApellNombre,    copia.Sexo,    copia.DNI,
copia.Nacimiento.Dia, copia.Nacimiento.Mes, copia.Nacimiento.Anio);

 getch();
 return 0;
}
}
```

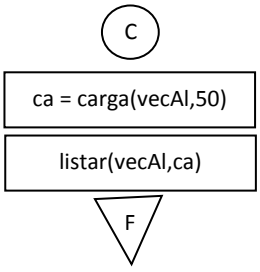
## 8. Estructuras con vectores

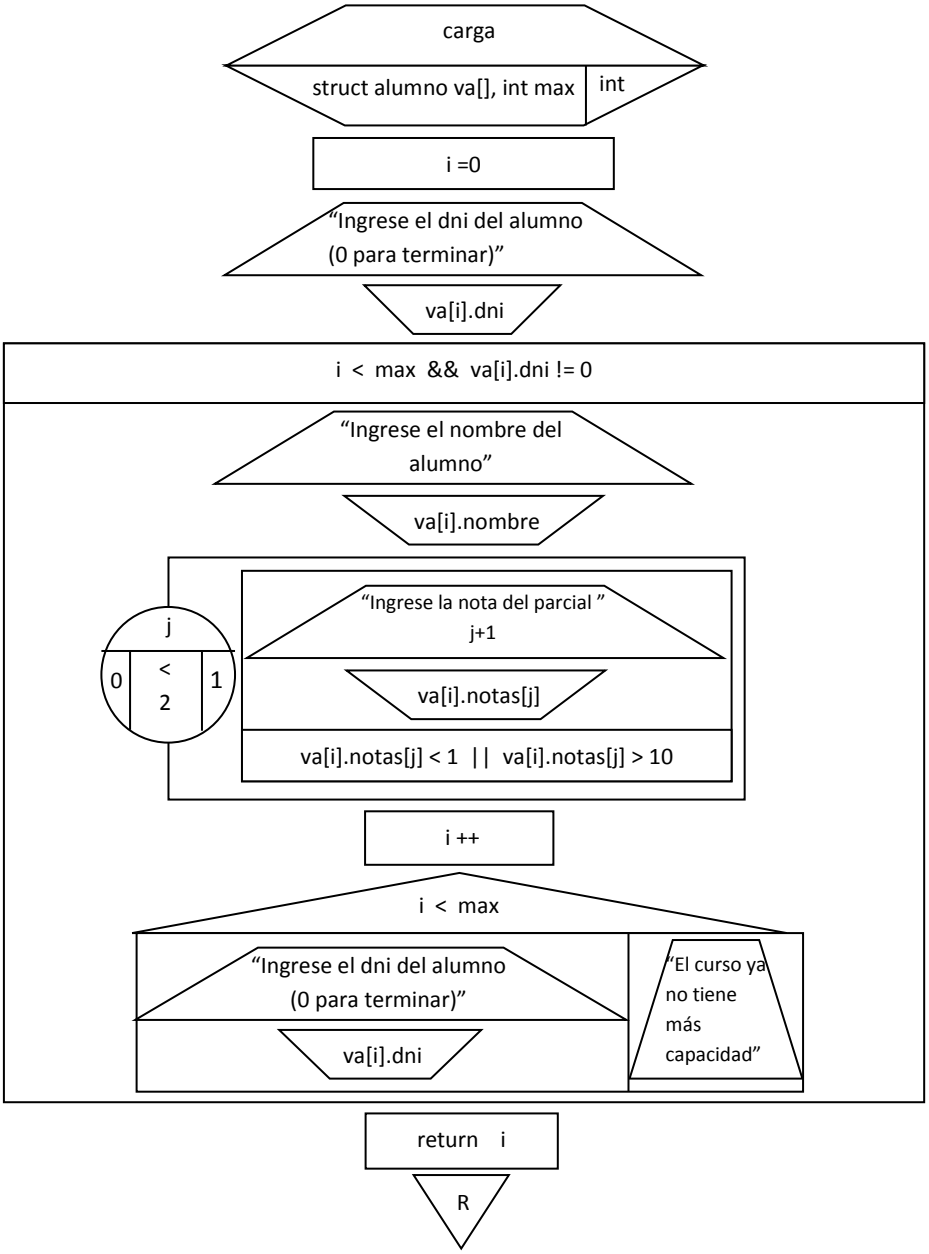
Una estructura puede contener cualquier otro tipo de dato dentro, incluyendo otras estructuras (como se vio en ejemplos anteriores) y también puede tener arrays (vectores y matrices). Por lo tanto, al momento de desarrollar el programa se debe prestar atención en donde colocar los subíndices ya que no debe confundirse un vector de estructuras con una estructura con un vector como campo.

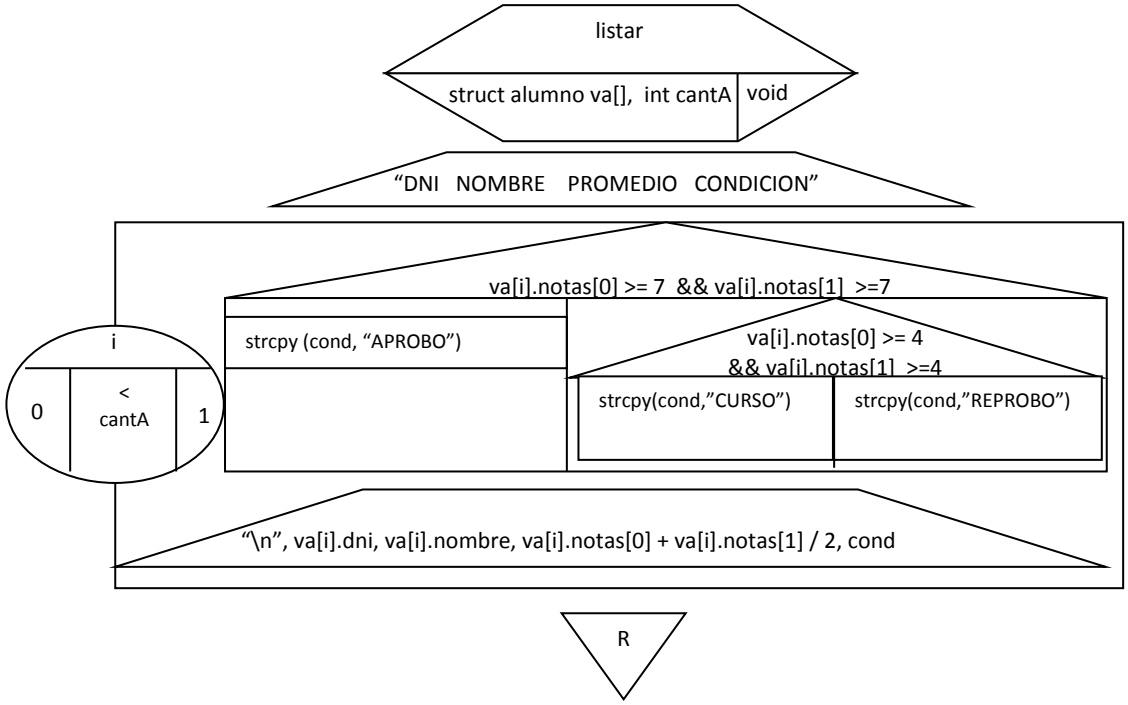
El siguiente ejemplo define la estructura alumno donde uno de sus campos es un vector con las notas de los dos parciales.

```
struct alumno
{
  char nombre [31];
  int dni;
  int notas[2];
};
```

Ejemplo 11: Ingresar los nombres, dni y notas de los alumnos de un curso. Se sabe que no son más de 50 personas (se finaliza con un dni igual a 0). Al finalizar el ingreso mostrar un listado con el promedio y la condición final de cada uno.




carga
struct alumno va[], int max
int
i =0
“Ingrese el dni del alumno
(0 para terminar)”
va[i].dni
i < max && va[i].dni != 0
“Ingrese el nombre del
alumno”
va[i].nombre
j
0
<
2
1
“Ingrese la nota del parcial ”
j+1
va[i].notas[j]
va[i].notas[j] < 1 || va[i].notas[j] > 10
i ++
i < max
“Ingrese el dni del alumno
(0 para terminar)”
va[i].dni
“El curso ya
no tiene
más
capacidad”
return i
R




```
#include <stdio.h>
#include <conio.h>
#include <string.h>

struct alumno
{
  char nombre [31];
  int dni;
  int notas[2];
};

int carga(struct alumno[], int);
void listar(struct alumno[], int);

int main()
{
    struct alumno vecAl[50];
    int ca;
    ca = carga(vecAl,50);
    listar(vecAl,ca);
    getch();
    return 0;
}

int carga(struct alumno va[], int max)
{
    int i=0,j;
    printf("Ingrese el dni del alumno (0 para terminar): ");
    scanf("%d", &va[i].dni);
    while (i<max && va[i].dni!=0)
    {
        printf("Ingrese el nombre del alumno:");
        getchar();
        gets(va[i].nombre);
        for (j=0;j<2;j++)
        {
            do
            {
                printf("Ingrese la nota del parcial %d: ", j+1);
                scanf("%d", &va[i].notas[j]);
            }while(va[i].notas[j]<1 || va[i].notas[j]>10);
        }
```

```
        i++;
        if (i<max)
        {
            printf("Ingrese el dni del alumno (0 para terminar): ");
            scanf("%d", &va[i].dni);
        }
        else
        {
            printf("El curso ya no tiene mas capacidad.\n");
        }
    }
    return i;
}

void listar(struct alumno va[], int cantA)
{
    int i;
    char cond[9];
    printf ("%10s%31s%9s%10s", "DNI","NOMBRE", "PROMEDIO" , "CONDICION");
    for (i=0;i<cantA;i++)
    {
        if (va[i].notas[0]>=7 && va[i].notas[1]>=7)
            strcpy(cond, "APROBO");
        else
            if (va[i].notas[0]>=4 && va[i].notas[1]>=4)
                strcpy(cond, "CURSO");
            else
                strcpy(cond, "REPROBO");

        printf      ("\n%10d%31s%9.2f%10s",      va[i].dni,va[i].nombre,      (va[i].notas[0]
+va[i].notas[1])/2.,cond);
    }
}
```

## 9. Otra forma de declarar las estructuras

La forma que se mostró hasta ahora para declarar una estructura es utilizando la palabra reservada struct seguido del nombre de la estructura y entre llaves sus campos. Por ejemplo, la estructura fecha que fue utilizada en ejemplos anteriores se define de la siguiente forma:

```
struct FECHA
{
   int Dia;
   int Mes;
   int Anio;
};
```

Al definir una estructura de este tipo vimos que al referirnos al nuevo tipo de dato en todos lados debemos poner struct FECHA, es decir siempre hay que poner la palabra reservada struct seguido del nombre de la estructura creada.

Pero existe una forma alternativa para declarar una estructura que elimina la necesidad de escribir siempre la palabra reservada struct para definir una variable del nuevo tipo de dato. Para ello se utiliza otra palabra reservada typedef declarando la estructura de la siguiente forma:

```
typedef struct
{
   int Dia;
   int Mes;
   int Anio;
}FECHA;
```

Esta forma de declarar la estructura hace que no se deba utilizar la palabra reservada struct al declarar una variable ya que FECHA por sí misma ya es un nuevo tipo de dato. La declaración de una variable entonces se hace de la siguiente forma:

```
FECHA fe;
```

Es decir, solo se pone el nombre del nuevo tipo de dato y luego el identificador de la variable a definir. De igual manera al escribir los parámetros de funciones no se debe poner la palabra struct. Esta forma alternativa reduce un poco el código fuente. Pueden utilizarse indistintamente cualquiera de las dos formas según preferencia del programador.

## *10. Ordenar un vector de estructuras*

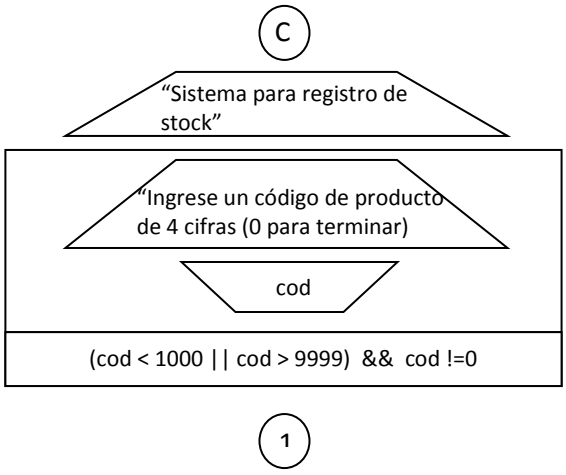
Para ordenar un vector de estructuras se puede utilizar cualquier método de ordenamiento. Anteriormente se vio el método por burbujeo en el cual se requería de realizar intercambios para ir ordenando los datos. En el caso de los vectores paralelos, se debían realizar varios intercambios para que la información no quede mezclada. La ventaja de los vectores de estructuras es que al definir un nuevo tipo de dato la variable auxiliar para el intercambio será directamente del tipo estructura permitiendo en una sola asignación copiar todos los sus campos. El ejemplo 12 muestra el desarrollo de un ejercicio donde se ordena un vector de estructuras.

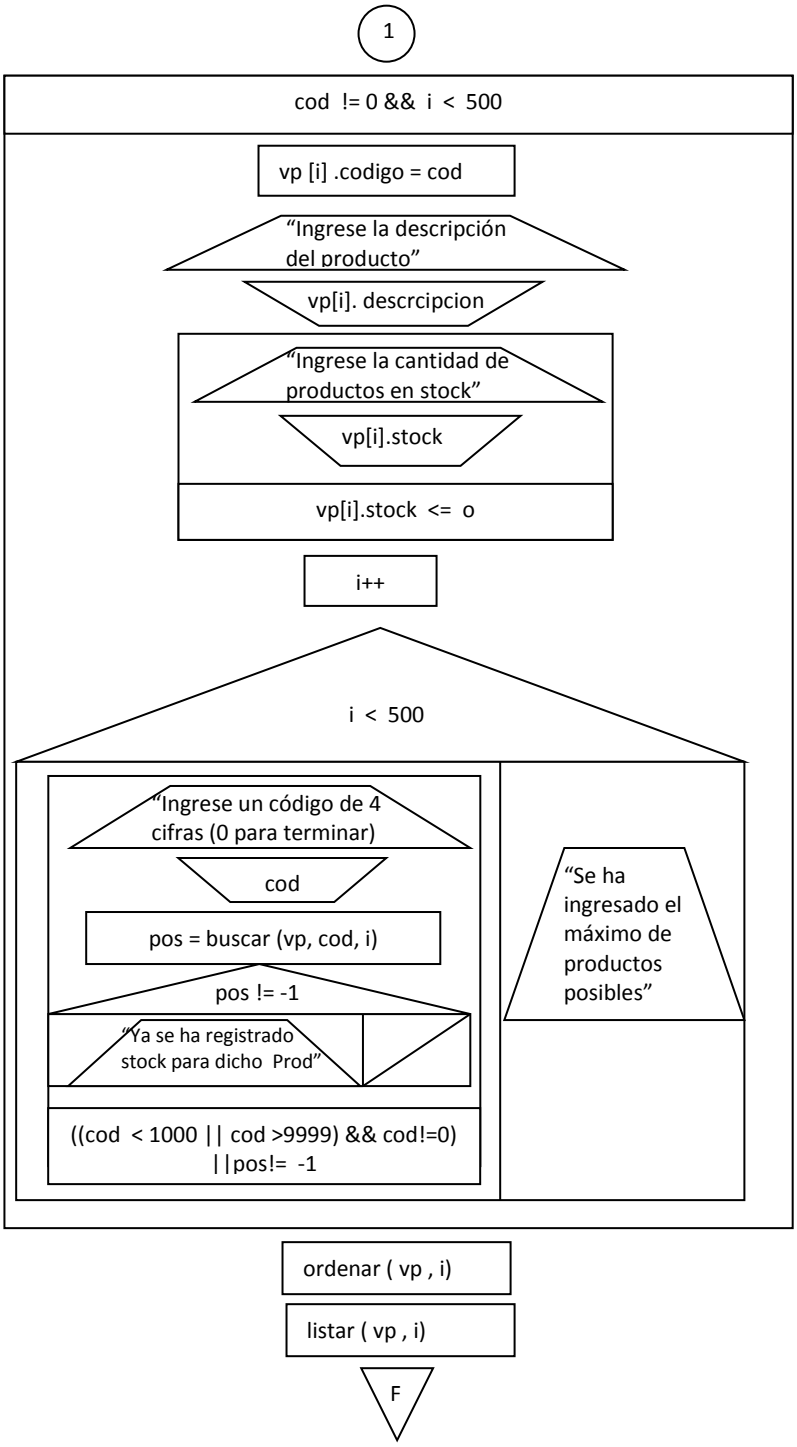
Ejemplo 12:

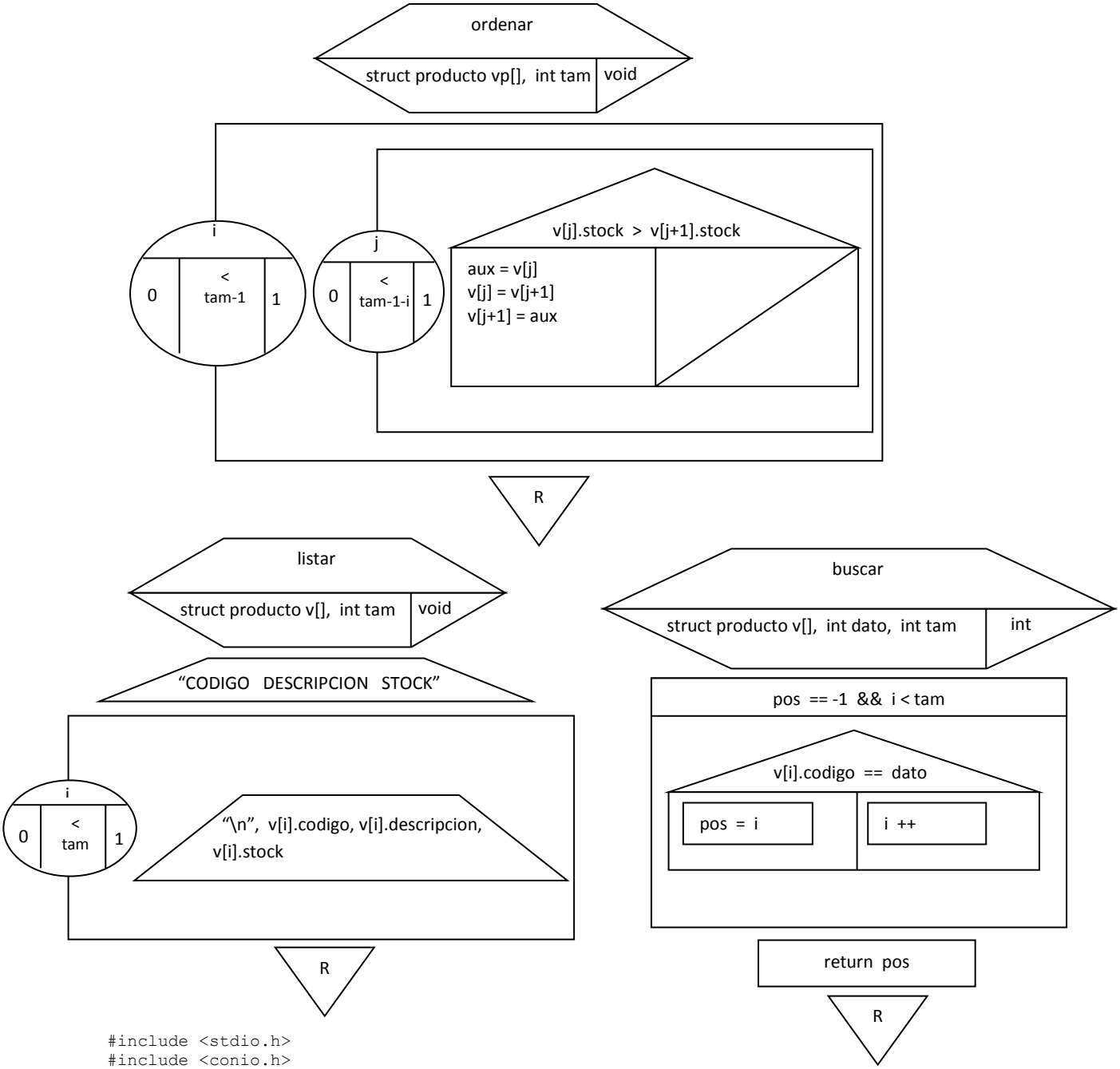
Realizar un sistema que permita hacer el inventario de productos en una fábrica. Para ello por cada por producto se debe ingresar

- Código (entero de 4 cifras)
- Descripción (texto de 20 caracteres máximo)
- Cantidad de productos en stock (entero mayor a 0)

No se sabe la cantidad total de productos, pero sí se sabe que no hay más de 500.
La carga de productos finaliza con un código de producto igual a 0.
Al finalizar mostrar en forma ordenada de mayor a menor por cantidad en stock los productos.




1
cod != 0 && i < 500
vp [i] .codigo = cod
"Ingrese la descripción del producto"
vp[i]. descrcipcion
"Ingrese la cantidad de productos en stock"
vp[i].stock
vp[i].stock <= o
i++
i < 500
"Ingrese un código de 4 cifras (0 para terminar)
cod
pos = buscar (vp, cod, i)
pos != -1
"Ya se ha registrado stock para dicho Prod"
((cod < 1000 || cod >9999) && cod!=0) ||pos!= -1
"Se ha ingresado el máximo de productos posibles"
ordenar ( vp , i)
listar ( vp , i)
F




```
#include <stdio.h>
#include <conio.h>

struct producto
{
    int codigo;
    char descripcion[21];
    int stock;
};

int buscar(struct producto[], int, int);
void ordenar (struct producto[], int);
void listar (struct producto[], int);

int main()
{
    struct producto vp[500];
    int cod, i=0, pos;
    printf ("SISTEMA PARA REGISTRO DE STOCK\n\n");

    do
    {
        printf ("Ingrese un codigo de producto de 4 cifras (0 para terminar):");
```

```
        scanf("%d", &cod);
    }while ((cod<1000 || cod > 9999) && cod!=0);

    while (cod!=0 && i<500)
    {
        vp[i].codigo = cod;

        getchar(); //limpia el buffer de teclado
        printf("Ingrese la descripcion del producto:");
        gets(vp[i].descripcion);

        do
        {
            printf("Ingrese la cantidad de productos en stock:");
            scanf("%d", &vp[i].stock);
        }
        while(vp[i].stock<=0);

        i++;
        if (i<500)
        {

            do
            {
                printf ("Ingrese un codigo de producto de 4 cifras (0 para terminar):");
                scanf("%d", &cod);
                pos = buscar(vp, cod, i);
                if (pos!=-1)
                    printf("Ya se ha registrado stock para dicho producto.\n");
            }while (((cod<1000 || cod > 9999) && cod!=0 ) || pos!=-1);

        }
        else
            printf ("Se ha ingresado el maximo de productos posible");
    }

    ordenar(vp, i);
    listar(vp, i);
    getch();
    return 0;
}

int buscar(struct producto v[], int dato, int tam)
{
    int i=0, pos=-1;
    while (pos==-1 && i<tam)
    {
        if (v[i].codigo == dato)
            pos =i;
        else
            i++;
    }
    return pos;
}

void ordenar (struct producto v[], int tam)
{
    int i,j;
    struct producto aux;

    for (i=0;i<tam-1;i++)
        for (j=0;j<tam-1-i;j++)
            if (v[j].stock > v[j+1].stock)
            {
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] =aux;
            }
}

void listar (struct producto v[], int tam)
{
    int i;
    printf ("\n%6s%21s%6s", "CODIGO", "DESCRIPCION", "STOCK");
    for (i=0;i<tam;i++)
        printf ("\n%6d%21s%6d", v[i].codigo, v[i].descripcion, v[i].stock);
}
```