HDC

C Programming

¿Qué es un puntero?

Muy buenas, en esta ocasión vengo a hablarles sobre algo muy importante en el lenguaje de programación C. Se trata de los **punteros**.

Un **puntero** es una **variable** que en lugar de contener un valor contiene la dirección una **dirección de memoria**, es decir que puede apuntar a otra variable. Esto tiene muchas aplicaciones, pero lo primero de todo es saber cómo se declara un puntero, y la sintaxis sería con el **asterisco** delante del nombre, como en la siguiente:

```
int *puntero;
```

En este caso estamos declarando un **puntero numérico entero**. Otra cosa importante es la **asignación de la dirección de memoria** del puntero.

“Woow. ¿Dirección de memoria?”

Claro, Manolo. Los punteros son especiales porque son una variable que almacena una dirección. La dirección de memoria donde se podrá guardar una variable del valor que elegimos anteriormente. En nuestro caso será un entero. **Cuando hablamos del puntero, hablamos de la dirección de memoria. Cuando hablamos de la variable o de la dirección a donde apunta, hablamos del valor int.**

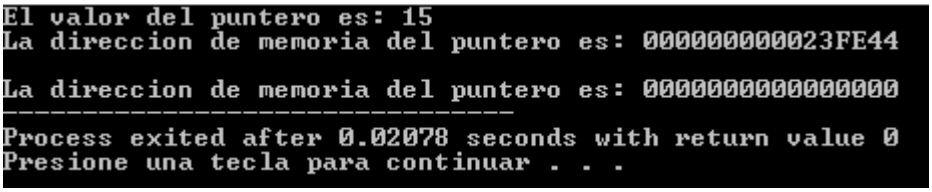
Esto quiere decir que si cambiamos el valor del puntero, perderemos dónde se encuentra el valor de la variable a la cual apunta nuestro puntero pero no se borraría ese valor. Claro, seria imposible de identificar dónde se encuentra en toda la memoria.

Veamos con un ejemplo a ver si nos despavilamos. La sintaxis es la siguiente:

```
1.      int numero = 0;
2.      int *puntero;
3.      puntero = &numero;
```

Aquí estamos creando una variable numérica y con un puntero estamos **apuntando a su dirección de memoria.** Esto lo hacemos anteponiendo "**&**" a la variable. Se puede comprobar el valor que toma el puntero imprimiéndolo en pantalla con el siguiente código:

```
1.      #include <stdio.h>
2.
3.      int main()
4.      {
5.          int numero = 15;
6.          int *puntero;
7.          puntero = &numero;
8.
9.
10.         printf("El valor del puntero es:
%d \n", *puntero);
11.         printf("La direccion de memoria del puntero
es: %p\n\n", puntero);
12.
13.         puntero = NULL;
14.
15.         printf("La direccion de memoria del puntero
es: %p", puntero);
16.
17.         return 0;
18.     }
```



Aquí hay varias cosas que hay que tener en cuenta:

1. Para imprimir el valor de un puntero por pantalla hay que escribirlo como ***puntero**.

2. Para escribir la dirección de memoria de un puntero tan solo hay que escribir el nombre de la variable (puntero, en nuestro caso) a la hora de usar la función printf() o la función que deseen.

3. El puntero se puede igualar a **NULL**, que apunta a la dirección de memoria 0.

Operaciones con punteros

Otra cosa importante de conocer es como se relacionan los punteros con las **operaciones**, tanto aritméticas como lógicas. En el caso de las operaciones **lógicas** es muy sencillo y **funciona como si estuviéramos comparando con una variable normal**, aquí podemos ver un ejemplo:

```
1.      if (numero == *puntero){
2.              printf("El valor del puntero es:
%d \n", *puntero);
3.          }
```

En el caso de las **operaciones aritméticas** modificar un puntero modificaría la variable y viceversa, se puede comprobar si añadís esto al código:

```
1.          numero = numero + *puntero;
2.
```

```
3.          *puntero = numero + *puntero;
```

Listas

Una vez sabido este creo que es el momento de explicar una de las aplicaciones más comunes que se le dan a los punteros, y es la de crear **listas.** Una lista es como un arreglo (array), solo que en lugar de tener un número fijo de datos, podemos ir metiendo datos y estos se van añadiendo a la memoria.

Para hacer esto lo primero que hay que hacer es declarar un tipo de dato nuevo en el que pondremos un puntero que apuntará al siguiente elemento de la lista.

```
1.          struct persona{
2.                  char nombre[20];
3.                  int DNI;
4.                  struct persona *siguiente;
5.          }
```

Si os fijáis aquí se ha declarado un arreglo de caracteres, esta es la forma de declarar un string en C, el primer valor del string se encuentra en la posición cero y la forma de printearlo es con “%s”, este ejemplo pide un string y modifica la primera letra.

```
1.      #include <stdio.h>
2.
3.      int main(){
4.          char palabra[5];
5.
6.          scanf("%s", &palabra);
7.          palabra[0] = 'H';
```

```
8.          printf("La palabra modificada es:
%s", palabra);
9.      }
```

Vamos a continuar haciendo la lista, para ello tenemos que declarar un puntero que sirva como el primero de la lista.

```
struct persona *primero, *ultimo;
```

Ahora vamos a crear un método que pueda usarse para añadir nuevos elementos a la lista:

```
1.      void anadir_elemento() {
2.            struct persona *nuevo;
3.
4.            // Asigna el espacio de memoria.
5.
nuevo = (struct persona *) malloc (sizeof(struct persona
));
6.            if (nuevo == NULL) {printf( "Fallo al añadir
nuevo elemento\n")};
7.
8.
9.            // Se introducen los datos para el nuevo
elemento
10.           printf("\nIntroduce los datos del nuevo
elemento:\n");
11.           printf("Nombre: "); fflush(stdout);
12.           scanf("%s"&nuevo->nombre);
13.           printf("DNI: "); fflush(stdout);
14.           scanf("%i"&nuevo->DNI);
15.
16.           // El campo siguiente va a ser NULL por ser
el último elemento de la lista
```

```
17.             nuevo->siguiente = NULL;
18.
19.             // Ahora metemos el nuevo elemento en la
lista.
20.             // Comprobamos si la lista está vacía con
primero=NULL
21.             if (primero==NULL) {
22.                 primero = nuevo;
23.                 ultimo = nuevo;
24.                 }
25.             // El nuevo elemento se mete al final de la
lista.
26.             else {
27.                     ultimo->siguiente = nuevo;
28.                     ultimo = nuevo;
29.             }
30.         }
```

En este código hay varias cosas que habría que comentar:

La función **malloc** asigna el número especificado de bytes, en este caso le asignamos el tamaño que ocupa nuestra estructura con la función **sizeof**. Para usar esta función hay que añadir la librería **<stdlib.h>**.

La función fflush sirve para vaciar el buffer, de una forma simple de entender, sirve para que no haya fallos entre los datos.

Lo siguiente que vamos a hacer es crear un método que usaremos para poder ver la lista entera:

```
1.      void mostrar_lista() {
2.
3.          // Se crea una nueva variable para poder
recorrer la lista
4.            struct persona *auxiliar;
5.            int i;
6.
7.            i=0;
8.            auxiliar = primero;
9.            printf("\nMostrando datos de la lista:\n");
10.     // Se recorre la lista con un bucle
11.           while (auxiliar!=NULL) {
12.                 printf( "Nombre: %s, DNI: %i\n",
13.                         auxiliar->nombre,auxiliar-
>DNI);
14.                 auxiliar = auxiliar->siguiente;
15.                 i++;
16.           }
17.           if (i==0) printf( "\nLa lista esta
vacia.\n" );
18.     }
```

Ya tan solo quedaría añadir un menú para que el usuario pueda interactuar con el programa, el código completo puede ser de la siguiente forma:

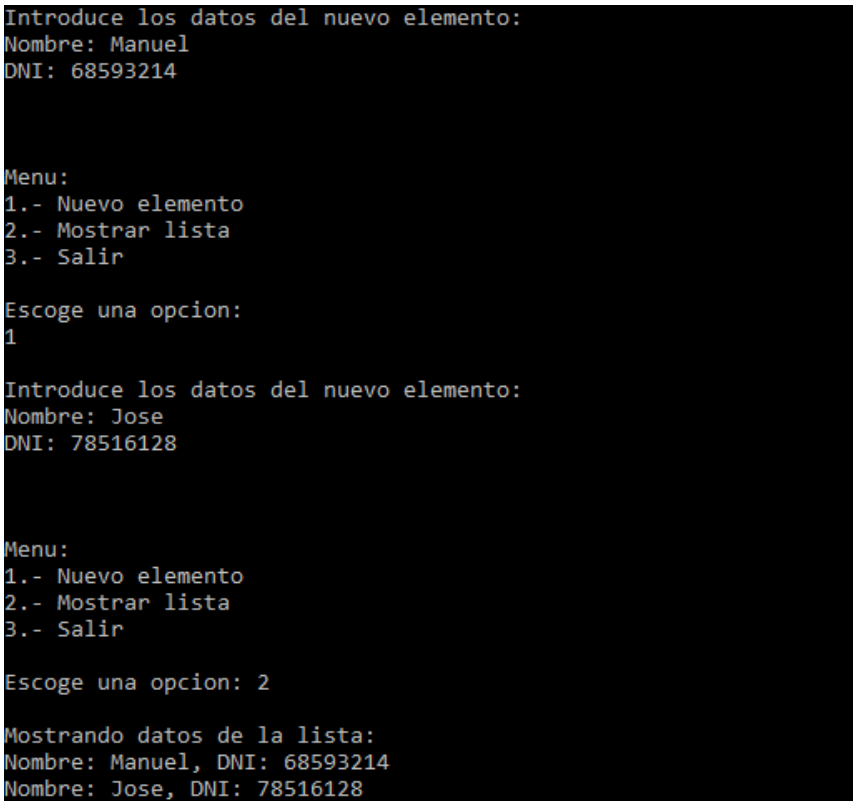
```
1.      #include <stdio.h>
2.      #include <stdlib.h>
3.
4.      struct persona{
5.          char nombre[20];
6.          int DNI;
7.          struct persona *siguiente;
8.      };
9.
10.     struct persona *primero, *ultimo;
11.
```

```
12.     void anadir_elemento() {
13.         struct persona *nuevo;
14.
15.         // Asigna el espacio de memoria.
16.
nuevo = (struct persona *) malloc (sizeof(struct persona
));
17.         if (nuevo == NULL) {printf( "Fallo al añadir
nuevo elemento\n");}
18.
19.
20.         // Se introducen los datos para el nuevo
elemento
21.         printf("\nIntroduce los datos del nuevo
elemento:\n");
22.         printf("Nombre: "); fflush(stdout);
23.         scanf ("%s",&nuevo->nombre);
24.         printf("DNI: "); fflush(stdout);
25.         scanf ("%i",&nuevo->DNI);
26.
27.         // El campo siguiente va a ser NULL por ser el
último elemento de la lista
28.         nuevo->siguiente = NULL;
29.
30.         // Ahora metemos el nuevo elemento en la
lista.
31.         // Comprobamos si la lista está vacía con
primero=NULL
32.         if (primero==NULL) {
33.             primero = nuevo;
34.             ultimo = nuevo;
35.             }
36.         // El nuevo elemento se mete al final de la
lista.
37.         else {
38.             ultimo->siguiente = nuevo;
39.             ultimo = nuevo;
40.         }
```

```
41.     }
42.
43.     void mostrar_lista() {
44.
45.         // Se crea una nueva variable para poder
recorrer la lista
46.           struct persona *auxiliar;
47.           int i;
48.
49.           i=0;
50.           auxiliar = primero;
51.           printf("\nMostrando datos de la lista:\n");
52.           while (auxiliar!=NULL) {
53.                 printf( "Nombre: %s, DNI: %i\n",
54.                         auxiliar->nombre,auxiliar-
>DNI);
55.                 auxiliar = auxiliar->siguiente;
56.                 i++;
57.           }
58.           if (i==0) printf( "\nLa lista esta
vacia.\n" );
59.     }
60.
61.     int main(){
62.         int opcion;
63.         while (opcion!=3){
64.             printf("\n\n\nMenu:\n");
65.             printf("1.- Nuevo elemento\n");
66.             printf("2.- Mostrar lista\n");
67.             printf("3.- Salir\n\n");
68.             printf("Escoge una opcion:
");fflush(stdout);
69.             scanf("%i", &opcion);
70.
71.
72.
73.
74.             switch (opcion) {
```

```
                    case 1:
                        anadir_elemento();
                        break;
                    case 2:
                        mostrar_lista();
                        break;
                    case 3:
                        break;
                    default:
                        printf( "Opcion no valida\n" );
                        break;
                }
            }
        }
```

La aplicación tendría este resultado por pantalla.



Aconsejo revisar varias veces el código para comprender bien como funciona, ya que al principio puede ser difícil de captar la idea.

Pueden seguirme en Twitter: @RoaddHDC

Contactarse por cualquier duda a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:

1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

También recomiendo que se unan al foro: underc0de.org/foro

Escrito por Rollth y revisado por Roadd.