LABORATORIO DE PROGRAMACIÓN III.

Unidad Nº 1: Punteros

. Concepto de Punteros.
. Punteros cercanos y lejanos.
. Punteros, representación de la memoria de la computadora.
. Definición e inicialización de punteros.
. Operadores * y &.
. Asignaciones de punteros.
. Aritmética de punteros.
. Comparación de punteros.
. Inicialización de punteros.
. Funciones de asignación dinámica de Turbo C.
. Operatoria elemental con Punteros cercanos.
. Llamadas a funciones y pasajes por referencia.
. Expresiones con punteros.
. Principios básicos de la aritmética de Punteros.
. Relación entre punteros y vectores.
. Punteros a cadenas de caracteres.
. Punteros a estructuras.
. Pasaje por referencia.
. Punteros a Punteros.
. Vectores de punteros.
. Punteros a funciones.

Un "puntero" es una variable que contiene una dirección de memoria. Normalmente, esa dirección es la posición de otra variable de memoria. Si una variable contiene la dirección de otra variable, entonces se dice que la primera variable "apunta" a la segunda. Por ejemplo, si una variable en la posición 1004 está apuntada por otra variable situada en la posición 1000, esa posición 1000 contendrá el valor 1004.

Dirección de memoria	Contenido
1000	1004
1001	
1002	
1003	
1004	
.	
.	

El correcto entendimiento y uso de los punteros es crítico, debido a las siguientes cuatro razones:

1. Los punteros proporcionan los medios por los cuales las funciones pueden modificar sus argumentos de llamada.
2. Los punteros se utilizan para soportar las rutinas de asignación dinámica de Turbo C.
3. El uso de punteros puede mejorar la eficiencia de ciertas rutinas.

4. A menudo se utilizan los punteros como soporte de ciertas estructuras de datos como las listas enlazadas y los árboles binarios.

Tamaño de los Punteros.

Existen dos tipos de punteros según su tamaño, con alcance de memoria diferente:

Puntero Cercano (near): ocupa 2 bytes en memoria. Es decir tiene 16 bits y por lo tanto puede contener 2 a la 16 direcciones de memoria diferentes, lo que le da un rango de alcance de 64 K posiciones. Esto es lo que corresponde con un segmento.

Puntero Lejano (far): Ocupa 4 bytes en la memoria. Esto permite alcanzar direcciones dentro del MB de memoria direccionable directamente. El formato con que se le entregan las direcciones es segmento – desplazamiento (número de segmento apuntado y desplazamiento a partir del inicio de dicho segmento.

Usos de los punteros.

Algunos usos frecuentes de los punteros y las ventajas que proporcionan son:

- Permiten el acceso a cualquier posición de memoria, para ser leída o escrita (en los casos en que esto sea posible). Por ejemplo, permiten el acceso directo a la memoria de video.
- Permite la transferencia de argumentos a las funciones **por referencia.** Hasta ahora se ha visto la transferencia de argumentos que las funciones tomaban en sus parámetros formales, **por valor.** En este caso, la función copia el valor transferido en una variable local (el parámetro formal) duplicando la memoria utilizada. Sólo se vio transferencia por referencia en el caso de los vectores; justamente porque lo que se estaba transfiriendo a la función era el nombre (del vector) que contiene la dirección de su inicio, es decir es un puntero (pero constante).
- Soportan las rutinas de asignación dinámica de memoria. Cuando es necesario solicitar memoria que no fue reservada al inicio del programa, se utilizan rutinas de asignación dinámica. Estas rutinas nos informan del lugar de la memoria otorgada a través de un puntero.
- Son el soporte de enlace que utilizan algunas estructuras avanzadas de datos, tales como las listas enlazadas, doblemente enlazadas, pilas, colas y árboles.
- Operan más eficientemente en los arrays que el modo de operación mediante subíndices.

Declaración del tipo puntero.

Si una variable va a contener un puntero, entonces tiene que declararse como tal. Una declaración de puntero consiste en un tipo base, un * y el nombre de la variable.

La forma general de declaración es:

Tipo * nombre;

Donde “tipo” es cualquier tipo válido de C (el tipo base del puntero) y “nombre” es el nombre de la variable puntero. El “ * ” es el operador de indirección que indica que la variable es de tipo puntero.

El tipo base es sumamente importante, pues tiene incidencia en el comportamiento del puntero frente a la aritmética de punteros.

Define el tipo de variables a las que puede apuntar el puntero. Técnicamente, cualquier tipo de puntero puede apuntar a cualquier lugar de la memoria, pero C asume que a lo que apunta un puntero es a un objeto de su tipo base. Además, toda la aritmética de punteros está hecha en relación a su tipo base, por lo que es importante declarar correctamente el puntero.

Ejemplo de declaración:

```
float x, *p;
```

En esta declaración se está indicando que x es una variable float y p es un puntero a float.

Los operadores de punteros.

Existen dos operadores especiales de punteros: **&** y *****.

1. El **&** es un operador monario (que sólo necesita un operando) que devuelve la dirección de memoria de su operando. Por ejemplo:

   m = &cuenta;

Pone en **m** la dirección de memoria de la variable **cuenta**. Esta dirección es la posición interna de la variable en la computadora. La dirección no tiene nada que ver con el valor de **cuenta**. Se puede pensar en el operador **&** como devolviendo "la dirección de". Por lo tanto, la declaración de asignación anterior significa "m recibe la dirección de cuenta".

Cuando se declara un puntero, éste contiene un valor desconocido (basura electrónica) y por lo tanto "no apunta a nada". Es lo que se llama "Puntero descontrolado". Es necesario entonces, inicializarlo con el valor adecuado.

Omitir la inicialización es uno de los errores más frecuentes que se producen con el uso de los punteros. Cuando se declara una variable y el puntero que la ha de apuntar, se realiza el proceso mental de relacionarlas, lo que no ocurrirá hasta que se lo declare explícitamente a través de una asignación con la dirección de la variable.

Ampliemos el ejemplo: Suponer que en la asignación anterior, la variable "cuenta" utiliza la posición de memoria 2000 para guardar su valor y que tiene un valor de 100. Entonces, ¿Qué significado tiene la sentencia del ejemplo?

Significa que después de la asignación m tiene el valor 2000.

2. El segundo operador de punteros, *****, es el complemento de **&**. Es un operador monario que devuelve el valor de la variable localizada en la dirección que sigue.
   Por ejemplo: si **m** contiene la dirección de memoria de la variable "cuenta", entonces:

   q = *m;

Pone el valor de **cuenta** en **q.** siguiendo con el ejemplo, **q** tendrá el valor 100 porque 100 está almacenado en la posición 2000, que es la dirección de memoria que se guardó en **m**. Se puede pensar en * como "en la dirección". En este caso, la sentencia anterior:

Significa que q recibe el valor en la dirección m.

El siguiente programa ilustra lo anterior:

```
# include <stdio.h>
main (void)
{ int cuenta, q;
  int *m;

  cuenta = 100;        /* a cuenta se le asigna 100*/
  m= &cuenta;          /* m recibe la dirección de cuenta*/
```

```
  q= *m;              /* a q se le asigna el valor de cuenta indirectamente a través de m */

  printf ("%d", q);   /* imprime 100 */

  return 0;
}
```

El programa anterior muestra un 100 en pantalla.

Tanto el operador & como * tienen mayor preferencia que todos los operadores aritméticos, excepto el menos monario, respecto del cual tienen la misma.

Las variables puntero deben apuntar siempre al tipo de datos correcto. Por ejemplo, cuando se declara un puntero a un tipo int, el compilador asume que cualquier dirección que mantenga apunta a una variable entera. Debido a que C permite asignar cualquier dirección a una variable puntero, el siguiente fragmento de código compila sin mensaje de error (aunque Turbo C mostrará un mensaje de advertencia), pero no produce el resultado deseado:

```
# include <stdio.h>
main (void)
{ float x, y;
  int *p;
  x = 100.123;
  p = &x;
  y = *p;
  printf ("% f", y); /*esto estará mal*/
  return 0;
}
```

Esto no asignará el valor de x a y. debido a que p se declara como un puntero a entero, sólo se transfieren 2 bytes de información a y, y no los 4 bytes que normalmente forman un número en coma flotante.

Asignación de punteros.

Como con cualquier variable, un puntero puede utilizarse a la derecha de una declaración de asignación para asignar su valor a otro puntero.

Un puntero puede ser asignado de 3 maneras:

1. A través de otro puntero.
2. Con la dirección de una variable.
3. Directamente con la dirección que deba contener.

1. La asignación de un puntero a través de otro puntero, se realiza en forma similar a la de cualquier otro tipo de variable.

```
# include <stdio.h>
# include <conio.h>
void main void
{ int *p , *q , A;
  clrscr ();

  q= &A;
  p = q;
/* Esto mostrará las direcciones contenidas en p y q. Serán las mismas */
```

```
printf ("%p %p", p, q);
getch ();
}
```

Luego de esta última sentencia, los dos punteros apuntan a A, y por lo tanto *p es equivalente a *q.

2. El caso de asignación de la dirección de una variable utiliza el operador & (en el tema **Los operadores de punteros**).

3. La asignación directa de una dirección se realiza entregándole un valor entero. Es necesario contemplar que el puntero almacena las direcciones en formato hexadecimal.

   Es decir, al realizar la asignación: p = 10; p contendrá 000AH.
   Lo habitual es entregarle la dirección expresada directamente en hexadecimal.
   Por ejemplo:

```
P = 0X0B75;
```

   Con lo cual p contendrá la dirección 0B75H.

Sin embargo, si bien la asignación anterior está clara y no presenta dudas, ambos tipos en la asignación son diferentes.

Mientras que p es un puntero a entero, 0X0B75 es un número entero. Este desapareamiento de tipos dará lugar a un "warning" (advertencia) o a un error, en el momento de compilación, dependiendo del compilador actuante.

Esta situación se resuelve mediante un "casting" en el momento de asignación, como se muestra a continuación:

```
Puntero = (tipo_base*) valor entero;
```

Ejemplo:

```
Int *p;
P = (int *) 0X0B75;
```

Comparación de punteros.

Los punteros soportan todos los operadores relacionales que actúan sobre las otras variables:

== != < <= > >=

Los punteros pueden ser comparados de tres formas posibles:

- Con otros punteros.
- Con direcciones de variables expresadas mediante el operador &.
- Con direcciones dadas directamente.

Ejemplo: Sean p y q punteros a entero, y A una variable entera.

- p < q Es una condición booleana en la que la dirección apuntada por p tiene menor valor numérico que la apuntada por q.

- p == &A Es una condición booleana en la que p apunta a A

- p != 0X0020 Condición booleana en la que la dirección contenida por p es diferente de 20H