UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Punteros

Una de esas cosas que la mayoría de los programadores en C encuentran difícil es el concepto de punteros. El propósito de este material es proporcionar una introducción a los punteros y su uso.

Los punteros son una de las poderosas herramientas que ofrece el lenguaje C a los programadores, sin embargo, son también una de las más peligrosas, en los programas de C, y, además, suele producir fallas muy difíciles de localizar y depurar que veremos un poco más adelante.

Definición: Un puntero es una variable destinada a contener una dirección de memoria.

Esta posición de memoria suele corresponder a la ubicación de otra variable, se dice entonces que el puntero "apunta" a la otra variable. Un puntero puede apuntar a un objeto de cualquier tipo como, por ejemplo, a una estructura o una función.

Los punteros permiten código más compacto y eficiente y, utilizándolos en forma ordenada dan gran flexibilidad a la programación.

El tamaño del puntero no depende del dato apuntado por él, sino de la arquitectura del compilador que estemos usando. En una máquina de 32 bits los punteros ocuparán 4 bytes, ya que deben tener el tamaño suficiente para almacenar direcciones de 32 bits.

La forma de declarar un puntero es:

```
tipo_dato    *nombre_puntero;
```

Donde **`tipo_dato`** es cualquier tipo de dato definido en C.

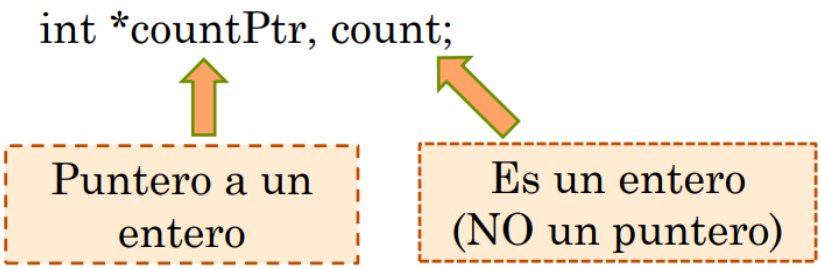
Ejemplo:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int     *puntero;
    char    *puntero2;
    float   *puntero3;

    return 0;
}
```

Otro Ejemplo:



El * no se aplica a todos los nombres de variables de una declaración. Cada puntero debe llevar su nombre precedido por *.

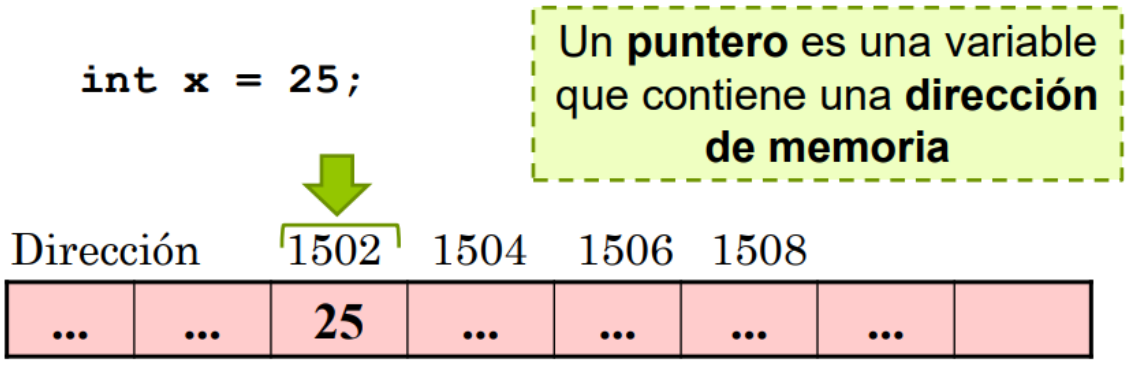
Algunas Conclusiones:

* Un puntero es una variable que contiene una dirección de memoria.

* Por lo general, una variable contiene un valor y un puntero a ella contiene la dirección de dicha variable.

* Es decir que la variable se refiere directamente a un valor mientras que el puntero lo hace indirectamente.

* **Una dirección de memoria y su contenido no es lo mismo. Ejemplo:**



La **dirección** de la variable x es 1502

El **contenido** de la variable x es 25

Operadores de Punteros

& -> Devuelve la dirección de memoria de su operando. Puede considerarse que equivale a “dirección de...”.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int x, *p;
7       p = &x; // p se carga con la dirección de x, es decir p apunta a x.
8
9       return 0;
10  }
11
```

Cuando se declara un puntero, este contiene un valor desconocido (basura electrónica) y por lo tanto no apunta a nada. Este es un problema común cuando uno declara punteros. El compilador de **Codeblocks**, o el **minGW** en caso de utilizar el puntero sin inicializarlo informa un **warning**.

* → Contenido de lo que apunta.

Ejemplo:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x,*p;
    p = &x; // p se carga con la dirección de x, es decir p
            // apunta a x entonces (*p) es el contenido de lo
            // apuntado por p, como p apunta a x, será entonces el
            // contenido de x.
            // *p es equivalente a utilizar la variable x.

   return 0;
}
```

El nombre del puntero es el identificador de la variable, el "*" es el operador de indirección que indica que la variable es de tipo puntero. El tipo es el tipo de variable apuntada por el puntero, denominado tipo base.

Ejemplo de cómo mostrar un elemento, la dirección de ese elemento, la dirección del puntero de distintas formas a través de punteros.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *pe,x=4;
    pe=&x;
    printf("Dirección de x : %p\t\t",&x);
    printf("Dirección de x : %p\n",pe);
    printf("Dirección de pe : %p\t",&pe);
    printf("Contenido del  puntero : %d\t", *pe);
    printf("Contenido de x : %d\n", x);

    return 0;
}
```

Uno de los casos más comunes donde se ve la relación entre estos dos operadores es la declaración y utilización de funciones:

```c
#include <stdio.h>
#include <stdlib.h>

void funcion( int* puntero);
// paso de una variable de tipo entera x dirección

int main()
{
    int a;
    a=6;
    funcion(&a); // la declaración de la función
                 // pide la dirección de una variable de tipo entero.
    return 0;
}
```

Muchas de las funciones estándares de C, trabajan con punteros, como es el caso del **scanf** o **strcpy**. Estas funciones reciben o devuelve un valor que es un puntero. Por ejemplo:

A **scanf** se le pasa la dirección de memoria del dato a leer

```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    char a;
    scanf ("%c",&a);
    return 0;
}
```

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Otro ejemplo:

```c
#include <stdio.h>
int main()
{   int *ptr;
    int dato=30;

    ptr = &dato;
    *ptr = 50;

    printf("Dato = %d\n", dato);

    return 0;
}
```

Declara un puntero a un entero

El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

```c
#include <stdio.h>
int main()
{   int *ptr;
    int dato=30;

    ptr = &dato;
    *ptr = 50;

    printf("Dato = %d\n", dato);

    return 0;
}
```

& es el **operador de dirección**: permite obtener la dirección de memoria de la variable que le sigue

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```c
#include <stdio.h>
int main()
{   int *ptr;
    int dato=30;

    ptr = &dato;
    *ptr = 50;

    printf("Dato = %d\n", dato);

    return 0;
}
```

No hay que confundir el * que aparece en **la declaración**

con

el **operador de indirección**

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Ejercicios:

1 - En base al código anterior:

```c
#include <stdio.h>
int main()
{   int *ptr;
    int dato=30;

    ptr = &dato;
    *ptr = 50;

    printf("Dato = %d\n", dato);

    return 0;
}
```

Cámbielo por
float * ptr

Ejecute y observe el resultado obtenido

2 - Ver que imprime el siguiente código:

```c
#include <stdio.h>
int main(void) {
    int a, b, c, *p1, *p2;

    p1 = &a;
    *p1 = 1;
    p2 = &b;
    *p2 = 2;
    p1 = p2;
    *p1 = 0;
    p2 = &c;
    *p2 = 3;
    printf("%d %d %d\n", a, b, c);

    return 0;
}
```

Visualización de Punteros

Puede utilizarse printf con la especificación de conversión **%p** para visualizar el valor de una variable puntero en forma de entero hexadecimal.

Ejemplo:

```
int Dato = 5, *PtrDato;

PtrDato = &Dato;

printf("%p\n", PtrDato);
```

0028FF1C

Aritmética de Punteros

A un puntero puede sumársele o restársele un entero, en el caso particular que este entero sea la unidad, tendremos las operaciones de incremento y decremento en posiciones de memoria.

Ejemplo:

```
p = p + 3;
p = p - 3;
p++;
p--;
```

Tiene especial importancia el tipo base apuntado.

La unidad de incremento o decremento de punteros no es el byte, sino la cantidad de bytes determinados por el tamaño del tipo base.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main()
6  {
7      float *p = 0X200;
8      p++;                    // p contendrá 200
9      p = p + 2;          // p contendrá 200CH
10
11     return 0;
12 }
13
```

Ocurre esto porque la variable float ocupa 4 bytes en memoria.

```
int main()
{
    int vec[10];
    int *p;
    p=vec;
```

vec contiene la dirección de inicio del vector por lo que vec es equivalente a &vec[0].

Pero como p fue asignado con la dirección contenida por vec, resulta que p también apunta a vec[0], a la misma dirección, es decir : vec es equivalente a p solo que uno es constante y el otro NO.

vec[i] hace referencia a la variable i-ésima en algún lugar del vector.

(p+i) apunta al elemento vec[i] mencionado precedentemente, y, por lo tanto:

p + i es equivalente a &vec[i]

Además

*(p+i) es equivalente a vec[i]

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Ejemplo de Manejo de Vectores con Punteros

```
1  #include<stdio.h>
2  void imprimirVector(int *vec, int ce);
3  void  cargarvector(int *p, int ce);
4
5  int  main()
6  {
7      int  vec[10],ce=5;
8      cargarvector(vec,ce);
9      imprimirVector(vec,ce);
10     return  0;
11 }
12
13 void  cargarvector(int* pv,  int  ce)
14 {
15     int  i=0;
16     printf("Cargar  %d  datos: ",ce);
17     for(i=0;  i<ce;  i++)
18     {
19         scanf("%d",pv);
20         pv++;
21     }
22 }
23
24
25 void imprimirVector(int *vec, int ce)
26
27 {
28     int i;
29     for(i=0; i<ce; i++)
30     {
31         printf("\n %d",*vec);
32                 vec++;
33     }
34 }
35
```

ACLARACIÓN: recordar que **VEC** un puntero constante que no se puede modificar donde está definido. Lo que se debe hacer es crear un puntero que contiene la dirección de vec.

En el ejemplo, cuando se pasa como parámetro a vec, lo que se está pasando es un puntero donde se copió la dirección de vec y que no es constante, por eso se puede usar aritmética de punteros en la función y **NO** donde se definió el puntero.

Vectores o Array

Un vector es un conjunto de variables contiguas de memoria del mismo tipo referenciadas por un nombre común e individualizadas mediante un subíndice numérico.

```
1   #include <stdlib.h>
2   #include<stdio.h>
3
4   int main()
5   {
6       int vec[5]; // declaración de un vector.
7       vec[2]=4; // asignación a uno de los elementos del vector.
8       vec[5] =3 + vec[2]; // asignación de otro int con una expresión.
9       scanf("%d", &vec[2]; // uso con scanf.
10
11      return 0;
12  }
13
14
```

El nombre del vector en realidad representa a **un puntero CONSTANTE que apunta a la dirección del primer elemento en ese bloque de memoria.**

Para conocer la dirección de un elemento del vector:

&vec[i] = vec + i

Donde:

i: representa el subíndice de la variable que se intenta acceder.

tipo: es el tipo de dato del elemento del vector.

Diferencia entre Typedef y Struct

Antes de iniciar el tema de punteros a estructuras, vamos a explicar la diferencia entre typedef y struct.

Typedef es una instrucción para renombrar un tipo de dato. Su formato es:

```
typedef <tipo_original> <nuevo_nombre_tipo>
```

Ejemplo:

```
typedef int entero; //hace que la palabra "entero" sea equivalente
al tipo int
```

Así en el programa:

```
entero num;
```

Será lo mismo que:

```
int num;
```

Si se pone antes de la declaración de un struct lo único que se está haciendo es mejorar la legibilidad del programa para cuando se necesite usar ese struct.

Se puede usar de ambas formas, pero el typedef ayuda a hacer más "bonitos" los programas.

Por ejemplo:

```
1   #include <stdlib.h>
2   #include<stdio.h>
3
4   struct alumno {
5   int edad, peso;
6   };
7
8   typedef struct{
9   int edad, peso;
10  } alumno;
11
12  int main()
```

De la 1ª forma, para usar ese struct en el programa se tendrá que declarar una variable de esta forma:

```
struct alumno alum;
```

De la 2ª forma, bastará con poner:

```
alumno alum;
```

Punteros a estructuras

Declaración:

```
struct tipo_struct * nom_punt;
```

Donde tipo_struct es un tipo de dato estructura definido previamente, y nom_punt es el identificador del puntero.

Ejemplo:

```
1   #include <stdlib.h>
2   #include<stdio.h>
3
4   struct fecha
5   {
6       int dia;
7       int mes;
8       int anio;
9   }; // estructura de tipo fecha
10
11
12  int main()
13  {
14      struct fecha hoy; // hoy es una variable estructura fecha;
15      struct fecha *p; // p es puntero a estructura fecha.
16      p = &hoy; //   Para que p apunte a hoy
17      return 0;
18  }
```

NO PUEDE ASIGNARSE UNA ESTRUCTURA DIRECTAMENTE A UN PUNTERO, SINO QUE ES NECESARIO EL OPERADOR & PARA INDICAR SU DIRECCIÓN.

Para acceder a los campos de una variable de tipo estructura fecha con punteros:

***p es equivalente a hoy.**

La referencia al campo mes de la estructura hoy se expresa como:

hoy.mes

Pero *p equivale a hoy (al valor de la variable hoy), podríamos expresar el campo mes de la estructura hoy, referenciándola a través del puntero p así:

(*p).mes

Pero hay una forma más simple y sintética Y **RECOMENDADA** de referenciar campos de estructuras a través de punteros y es el símbolo -> que da el significado "que apunta a…"

De esta manera podemos decir que:

p->mes equivale a **(*p).mes**

Ejemplo:

```
1   #include <stdlib.h>
2   #include<stdio.h>
3   typedef struct
4   {
5       char nombre[20];
6       int nota;
7   } t_alumno;
8
9   int main()
10  {
11      t_alumno alu= {"manuel",9};
12      t_alumno *p;
13      p=&alu;
14      printf("nombre : %s ",p->nombre);
15      printf("nota : %d",p->nota);
16      return 0;
17  }
```

Si antes del campo al que se accede está el dato tipo estructura va el '.'

pero si hay un puntero se usa '->'.

El typedef en este caso lo usamos para redefinir la estructura.

Recursos adicionales:

En el siguiente enlace encontrarás un video explicando estos conceptos:

https://www.youtube.com/watch?v=OgX4vdtkkHQ

https://www.youtube.com/watch?v=QV_6_aKH9G8

Referencias:

[1] B. W. Kernighan y D. M. Ritchie, El lenguaje de programación C, Pearson Educación, 1991.

[2] H. M. Deitel y P. J. Deitel, Cómo programar en C/C+, Pearson Educación, 1995.