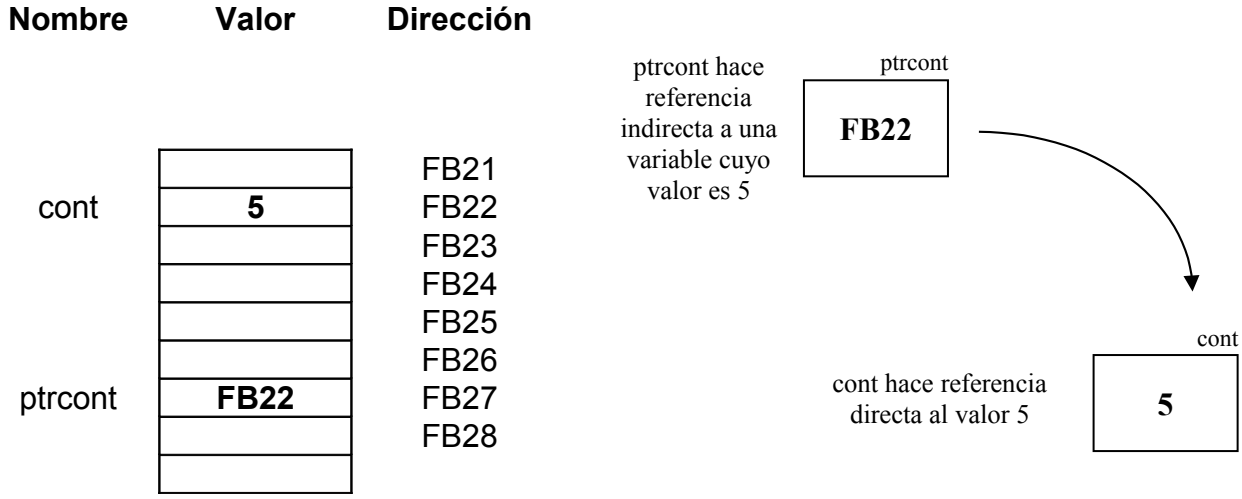
PUNTEROS

INTRODUCCION

Una de las características más poderosas del C, es el *puntero* o *apuntador*. Los punteros permiten simular las llamadas por referencia y crear y manipular estructuras de datos dinámicas, que pueden crecer y encogerse en tiempos de ejecución, como las pilas, árboles y listas. Nos introduciremos ahora en los conceptos básicos de los punteros.

DEFINICION DE PUNTEROS

Los punteros son variables cuyos valores son direcciones de memoria. Por lo general, una variable contiene un valor específico, un entero, un flotante, un carácter, etc. En cambio, una variable puntero contiene la dirección en memoria de una variable que contiene un valor específico. De esta manera, mediante el nombre o el identificador de una *variable*, hacemos referencia directa a un valor. En cambio, mediante el nombre o identificador de un *puntero*, hacemos referencia indirecta a un valor. Al proceso de referenciar a un valor a través de un puntero, se le llama *indirección*.



OPERADORES CON PUNTEROS

Debido a que los punteros son variables cuyos valores son direcciones de memoria, para utilizarlos de manera adecuada e implementar lo graficado, necesitaríamos almacenar en el puntero la dirección de memoria de la variable y recuperar el valor al que esta apuntando. Para Implementar lo explicado, vamos a utilizar dos operadores :

& → Operador de Dirección (o de referencia)

devuelve la dirección donde se almacena el operando

***** → Operador de Indirección (o de desreferencia)

devuelve el valor del objeto que apunta

Asi sería el código:

```
   int cont=5;

   int *ptrcont;

   ptrcont = &cont;

   printf(" Variable contador = %d y esta en %x →   referencia
directa", cont, &cont );

   printf(" Variable contador = %d y esta en %p →   referencia
indirecta", *ptrcont , ptrcont );
```

Llamada a Funciones con paso de parámetros

por puntero o por referencia

```
   #include <stdio.h>

   void cuboporpuntero ( int *ptrnro );      /* Prototipo */

   int main()
   {
      int numero = 5;
      printf ("El valor original de numero es %d", numero);
      cuboporpuntero ( &numero );
      printf ( "El nuevo valor de numero es %d ", numero);
      return 0;
   }

   void cuboporpuntero ( int *ptrnro )     /*cuerpo de la funcion*/
   {
      *ptrnro = *ptrnro * *ptrnro * *ptrnro;
   }
```

```
   # include  <stdio.h>

   void swap ( int *x, int *y ) ;      /* Prototipo */

   int main ( )
   {
      int var1 = 3;
      int var2 = 5;
      printf ("El valor original de la primera variable es :  %d \n", var1);
      printf ("El valor original de la segunda variable es :  %d \n", var2);
      swap ( &var1, &var1 );           /*llamada a la función intercambio*/
      printf ("Despues de llamar a intercambio, el valor de las variables es :
\n\n" );
```

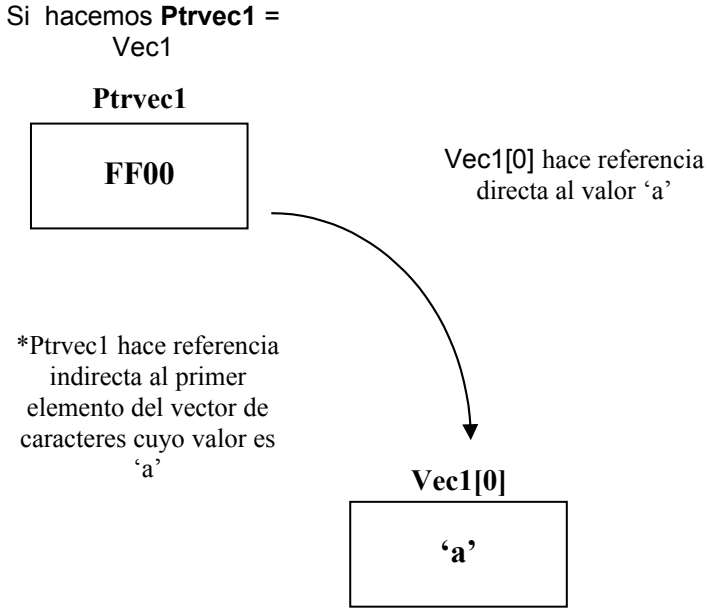
```
   printf ("El valor  de  la primera variable es :  %d \n", var1);
   printf ("El valor  de  la segunda variable es :  %d \n", var2);
   return 0;
}

void swap ( int *x, int *y )
{
   int tmp;
   tmp = *x;
   *x = *y;
   *y = tmp;
}
```

ARITMÉTICA DE PUNTEROS

Se puede realizar un conjunto limitado de operaciones con los punteros y en general el resultado de estas operaciones solo sigue las reglas de la aritmética convencional si es un puntero a un arreglo de caracteres o string en donde el objeto ocupa solo 1 byte en los otros caso en donde el objeto al que apunta es un entero o un flota no se cumplen estas reglas. Por ejemplo cuando a un puntero se le suma o se le resta un entero no aumenta o disminuye en dicho entero si no, en el número de veces del tamaño del objeto al que se hace referencia.

Nombre	Valor	Dirección
Vec1[0]	'a'	FF00
Vec1[1]	'b'	FF01
Vec1[2]	**'c'**	FF02
Vec1[3]	**'d'**	FF03
Vec1[4]	**'e'**	FF04



Si ahora incrementamos en 2 el puntero

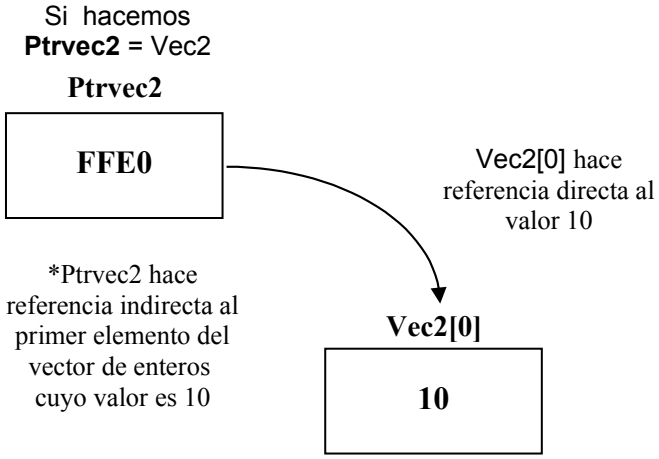
Ptrvec1 = Ptrvec1 + 2 → FF00 + 2*1 (cantidad de bytes que ocupa un caracter)

Ptrvec1 ahora vale FF02 y apunta a 'c'

En cambio tenemos un puntero a un arreglo de tipo entero

Como **int Vec2[5]**

Nombre	Valor	Dirección
Vec2[0]	10	FFE0
Vec2[1]	20	FFE2
Vec2[2]	30	FFE4
Vec2[3]	40	FFE6
Vec2[4]	15	FFE8



Si ahora incrementamos en 2 el puntero

Ptrvec2 = **Ptrvec2 + 2** → FFE0 + 2*2 (cantidad de bytes que ocupa un entero)

Ptrvec2 ahora vale FFE4 y apunta a 30

Uso del calificador const con apuntadores

Si una variable no se modifica (o no debería modificarse) en el cuerpo de la función a la cual se le pasa esta variable, en ese caso se declara como constante, utilizando el calificador const.

El calificador const puede utilizarse para reforzar el principio del menor privilegio. Reduciendo el tiempo de depuración y efectos colaterales indeseados, lo que hace un programa mas fácil de usar y mantener.

Existen tres casos:

1-Puntero no constante a dato constante

```
const int numero = 10;  /* Dato constante */

int *punt = &numero;  /* Puntero no constante */

*punt = 20;   /* Error: No se puede modificar el dato donde apunta punt, o sea, a numero */
```

2-Puntero constante a dato no constante

```
int numero1 = 10;

int numero2 = 20;

int *const punt = &numero1;

*punt = 5;   /* Se puede modificar el dato donde apunta punt, o sea, a numero1 */

punt = &numero2; /* Error: No se puede modificar el contenido de punt, o sea, su direccion*/
```

3-Puntero constante a dato constante

```
int const numero1 = 10;

int const numero2 = 20;
```

```
int *const punt = &numero1;

*punt = 5;   /* Error: No se puede modificar el dato donde apunta punt, o sea, a numero1 */

punt = &numero2; /* Error: No se puede modificar el contenido de punt, o sea, su dirección*/
```

Trabajo Práctico Nº 12

Completar el siguiente programa.

```
void intercambia (int *ptrnro1, int *ptrnro2)
{
      int aux=  *ptrnro1;
      *ptrnro1= *ptrnro2;
      *ptrnro2= aux;
}

void ordena_burbuja (int *const arreglo, const int tam)
{     void intercambia (int *, int *);              /* prototipo */
      int pasada;
      int j;
           for (pasada = 0; pasada  < tam-1; pasada++)
               for (j = 0; j < tam-1-pasada; j++)
                   if (arreglo[j] < arreglo[j+1])
                        intercambia (&arreglo[j], &arreglo[j+1]);
}

/* definir constante tam */

void main ()
{
      -- declaración de variables a utilizar
      -- ingreso del arreglo
      -- llamado a función
      -- impresión de resultados
}
```