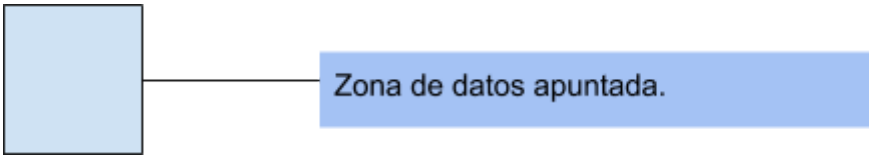
apuntes de punteros.

Concepto de puntero (pointer)

Zona de memoria situada en el HEAP a la que se le dá un nombre y almacenará la información de dirección de memoria a la que referencia una variable.



Los operadores de puntero existentes son 2: el operador de contenido, representado por un *
y el operador que indica la asignación de dirección de memoria, a este lo representaremos por el símbolo ampersand o la & inglesa.

Lo primero para poder definir un puntero es indicarle el tipo de datos necesario, cada puntero estará asociado a un tipo de datos específico y contendrá datos de ese tipo especifico, no obstante, existe una excepción a esta regla y es el puntero void.

Void es un puntero sin tipo (con tipo indefinido) o de todos los tipos posibles, es decir, es el único tipo de puntero que va a poder cambiar información entre tipos. Lo trataremos más adelante.

Uso de punteros

Para poder utilizar los punteros, primero tenemos que declararlos, se declaran como cualquier variable pero en su declaración se incluye un * inicial. Como ejemplo:

int *a; puntero a entero
char *b puntero a carácter

tras su declaración deberemos inicializar su uso, para ello, haremos que el puntero comience a apuntar a alguna dirección útil. Imaginemos que tenemos un programa con estos datos:

int a:
int *b;

a=30;
b=&a;

Ahora el puntero b apunta a la dirección de a, con lo que cualquier cambio en el contenido de b cambiará lo existente en a

Lo que hacemos es que apuntamos b a la posición que está ocupando el valor a en la memoria. esto va a ser útil cuando tenemos que manejar y devolver muchos datos en una función para evitar tener que ocupar mucha memoria o devolver estructuras muy grandes o complejas.

si no se asigna al puntero una variable estática, el uso del mismo se hace mediante la asignación dinámica de memoria. Para ello, se utilizará el comando malloc de c. Su sintaxis es la siguiente:

b = (*int) malloc(n*sizeof(char));

En este caso, el primer tipo *int es el tipo de datos al que va a pertenecer el puntero. lo siguiente es la propia orden malloc que recibirá como parámetro el número de bytes que se va a reservar para el puntero. Para realizar esta operación, disponemos del operador sizeof(tipo de dato).

Utilización de punteros

Los punteros son muy útiles para usarlos como arrays, en el caso de arrays simples, los elementos vienen dados por la longitud del tipo de datos. Si quiero por ejemplo, asignar valor para cinco enteros, tendré que hacerlo de la forma siguiente suponiendo que num sea puntero a entero:

int *num;

num = (*int) malloc (5* sizeof(int))

De esta forma, podríamos almacenar 5 números enteros como si estuviesemos manejando un array.

num[0]=1;
num[1]=3;

y operar con esos valores.

El uso de las tablas de dos dimensiones es prácticamente igual, salvo que hay que contar con que tenemos filas y columnas. un ejemplo de esto sería el siguiente:

int **num;

num = (int **)malloc(filas*sizeof(int*));

y para las columnas después tenemos que volver a aplicar la orden.

num[i] = (int*)malloc(columnas*sizeof(int));

luego se utilizará como si fuese un array normal.

Liberar memoria.

Al finalizar el uso de la memoria utilizada por el puntero, tenemos que limpiar la memoria para evitar que haya memory leacks o pérdidas de memoria. para ello, se utilizará la orden free con el nombre del puntero.

En el caso anterior, la orden free limpiará cada una de las dimensiones del array.

free (num[x]);

Punteros a funciones

Un puntero a función es una variable que almacena la dirección de una función. Esta función puede ser llamada más tarde, a través del puntero. Este tipo de construcción es útil pues encapsula comportamiento, que puede ser llamado a través de un puntero. Veamos cómo funciona mediante un ejemplo sencillo que crea un puntero a una función de imprimir y lo invoca:

```
1  #include <stdio.h>
2  void imprime()
3  {
4      printf("Imprimiendo un message\n");
5  }
6  int  main()
7  {
8      void (*ptr_funct)(void)=imprime;
9      ptr_funct(); //Llama a imprime
10     return 0;
11 }
```

Los punteros a funciones también se usan para realizar funciones de retrollamada (callback, en inglés). El siguiente trozo de código muestra un ejemplo donde una

función recibe como parámetro un puntero a función, el cual invoca cuando ha terminado su ejecución. También muestra las direcciones de memoria que ocupan las funciones, que se imprime con la opción %p de la función printf.

```
1   #include <stdio.h>
2   void imprimiendo_hola(int x)
3   {
4      printf( "[*HOLA__] Hola número %d\n", x);
5   }
6
7   void imprimiendo_adios(int x, void (*ptr_func)())
8   {
9      printf( "[*ADIOS_] Adios número %d\n", x);
10     if(ptr_func!=NULL)
11     {
12       ptr_func();  //Usando el puntero para retrollamar
13     }
14  }
15
16  void func_call_back()
17  {
18     printf( "[*RETRO_] Retrollamada llamada\n");
19  }
20
21  int main()
22  {  // Definimos dos punteros a funciones
23     void (*ptr_func_1)(int)=NULL;
24     void (*ptr_func_2)(int, void (*call_back_func)() )=NULL;
25     //Usamos el primero
26     ptr_func_1 = imprimiendo_hola;
27     printf("[*MAIN__] La dirección del primer puntero is %p\n",ptr_func_1);
28     ptr_func_1(3);
29     //Usamos el segundo
30     ptr_func_2 = imprimiendo_adios;
31     printf("[*MAIN__] Using a callback función \n",ptr_func_2);
32     ptr_func_2(3,func_call_back);
33     return 0;
34  }
```