**UNIDAD 8 - PUNTEROS EN LENGUAJE C**

CONCEPTO DE PUNTERO

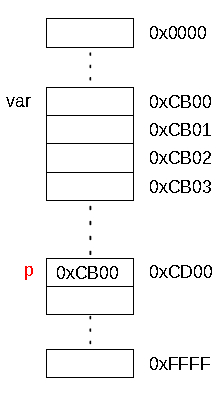
Los punteros son variables, y como tales tienen nombre, tipo, valor y una dirección donde almacena ese valor.

Lo particular que tienen los punteros es que el valor, es una dirección de memoria correspondiente a otra variable.

Como todas las variables, antes de usarse deben definirse.

Solo puede tener tres tipos de valores: 0, NULL o una dirección de memoria.

CONCEPTO DE DIRECCIÓN



Se dice que los punteros "apuntan" a una variable cuando su valor es la posición de memoria de esa variable.

En el ejemplo se dice que p

apunta a var.

En los gráficos de memoria usaremos letras rojas para diferenciar punteros de variables comunes

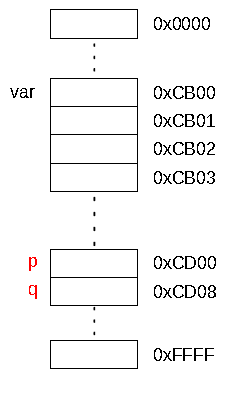
DECLARACIÓN DE PUNTEROS



Para declarar un puntero se usa un asterisco (\*) delante del nombre de la variable (o sea el puntero).

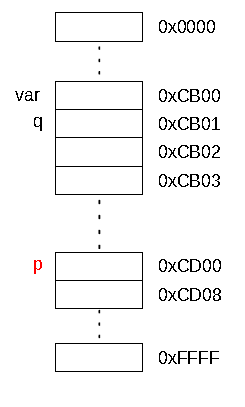
El tipo (en este caso int) significa que apunta a una variable que es de tipo int. Los punteros pueden ser de cualquier tipo.

Siempre deben apuntar a variables del mismo tipo al que fueron declarados.

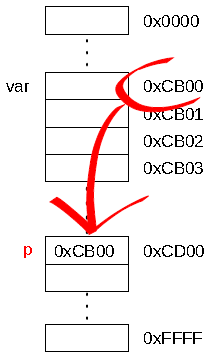


Cuando se declaran dos o más punteros en una misma sentencia, cada nombre debe llevar su asterisco.





En este caso, p es un puntero a char, pero q es simplemente un char.

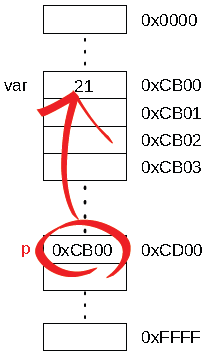


OPERADOR DE DIRECCIÓN (&)



El operador de dirección también llamado operador de referencia es un operador unario (& ampersan), que se coloca delante de una variable y devuelve la dirección de la misma.

Independientemente de que var tenga asignado o no un valor, el puntero apunta a esa posición de memoria, y se puede acceder a la misma para leer o cargar valores.

OPERADOR DE DESREFERENCIA (\*)

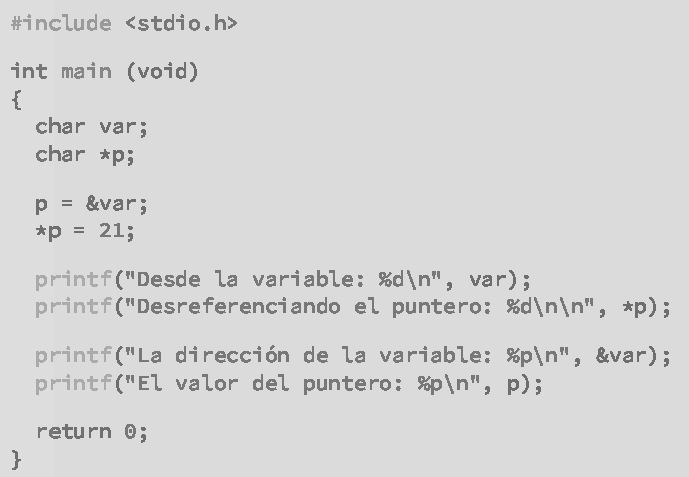
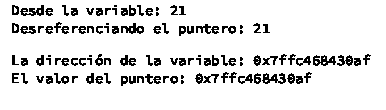


El operador de desreferencia también llamado operador de indirección es un operador unario (\* asterisco), que se coloca delante de un puntero.

Permite acceder indirectamente al contenido de la dirección de memoria apuntada

Se dice desreferenciar el puntero.

Puede ser tanto para cargar valor o usarlo en alguna expresión.



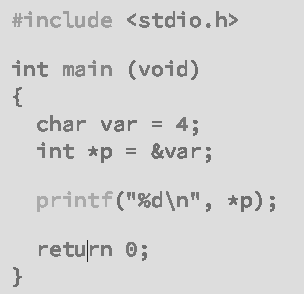
ASIGNACIONES VÁLIDAS

Como se dijo, los punteros solo pueden tener tres tipos de valores: 0, NULL o una dirección de memoria.

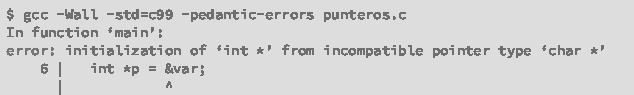
NULL es una constante definida en stdio.h (en realidad en stddef.h pero se incluye en la primera).

El 0 (cero) es el único entero que puede ser asignado a un puntero.

Las direcciones, solo pueden ser asignadas mediante el operador de dirección (&) u otros punteros.



NO se pueden hacer asignaciones de punteros de tipos incompatibles.



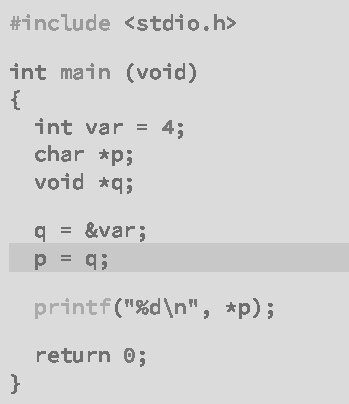
Solo se puede asignar un puntero a otro si ambos son del mismo tipo con la excepción de los punteros de tipo void.

El tipo void se puede usar en punteros, y significa que no es ningún tipo en particular.

Se puede asignar cualquier puntero a un puntero void.

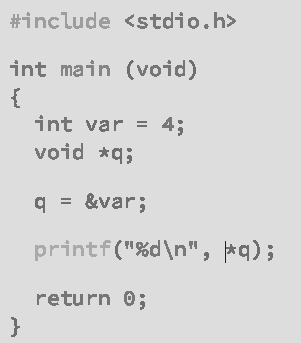
Se puede asignar un puntero void a cualquier tipo de puntero.

PUNTERO VOID

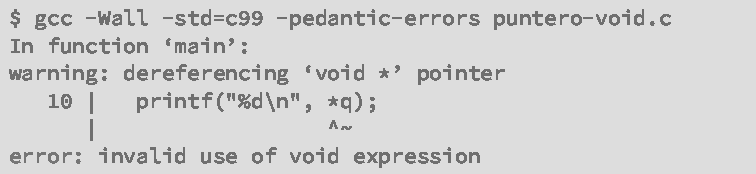


(q = &var) la dirección de un entero a un tipo void

(p = q) un puntero void a un puntero char



No se puede desreferenciar un puntero void.

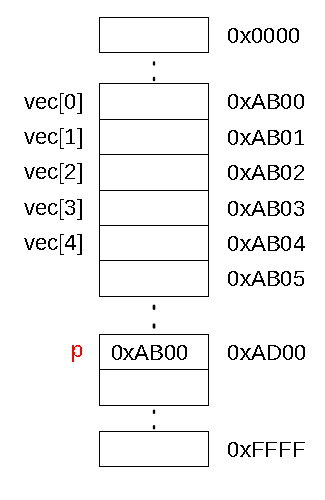
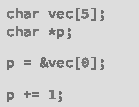


ARITMÉTICA DE PUNTEROS

Los punteros pueden ser operandos para operaciones de asignación, sumas y restas.

Incluyendo operadores de pre y post incremento y decremento (++, --) y de incremento y decremento y asignación (+=, -=).

No se pueden usar los operadores de división o multiplicación.



Los incrementos en el puntero pueden aprovecharse para recorrer el arreglo.

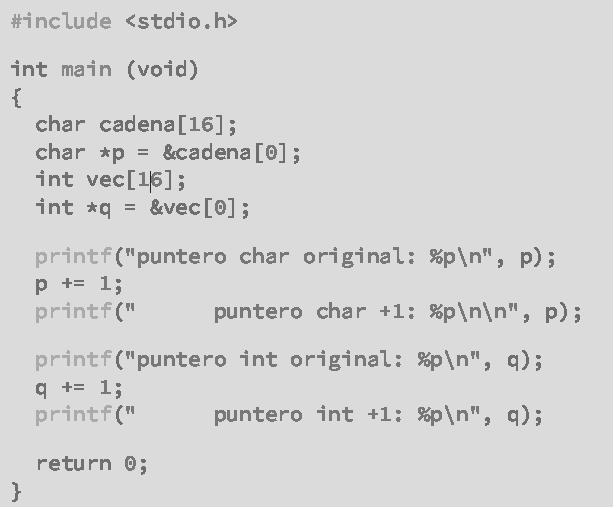


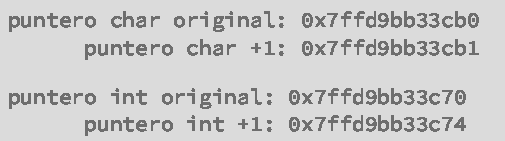


La línea incrementa en 1 del puntero a

char pasando por ejemplo de 0xAB00 a 0xAB01.

Sin embargo, si el puntero (y el arreglo) es de tipo int la misma línea provocaría que el puntero pase de 0xAB00 a 0xAB04.

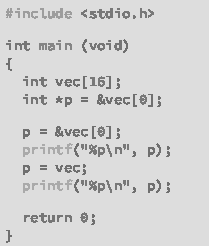




RELACIÓN ENTRE PUNTEROS Y ARREGLOS

Los punteros y los arreglos están íntimamente relacionados, y en ocasiones pueden ser intercambiados.

Esto es debido a que el nombre del arreglo (solo el nombre sin los corchetes) es igual a la dirección de memoria del primer elemento.



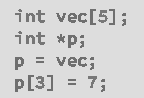
Después de la línea p = &vec[0]; se pueden acceder a los elementos desreferenciando el puntero y un desplazamiento, por ejemplo:



asigna un 7 al elemento vec[3]

Esto se conoce como notación puntero/desplazamiento,

también se puede hacer:



También es posible usar el puntero con notación puntero/ índice.

Esto será útil en las funciones.

**PUNTEROS EN LENGUAJE C (SEGUNDA PARTE)**

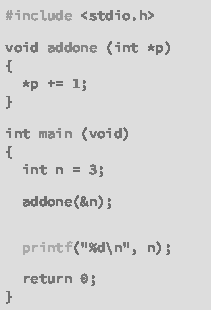
IMPLEMENTACIÓN DE LLAMADAS A FUNCIÓN POR REFERENCIA

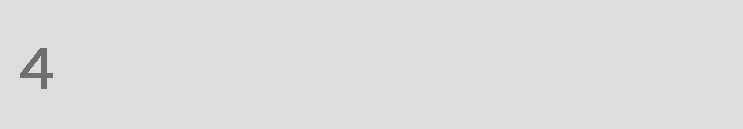
Como se dijo, en los lenguajes en general, los llamados a funciones se pueden hacer pasando variables por valor o por referencia.

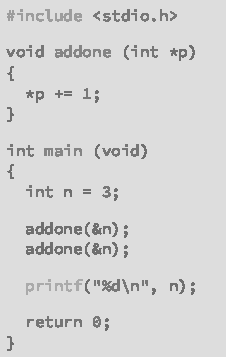
Sin embargo, en C los llamados son siempre por valor.

Pero se pueden implementar llamadas por referencia usando punteros, pasando la dirección de la variable a modificar en la función.

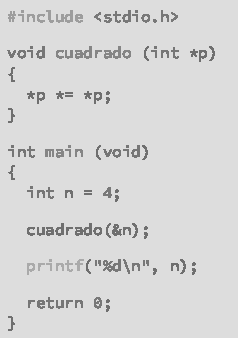
PASAJE DE PUNTEROS A FUNCIONES





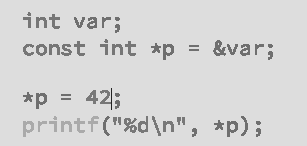


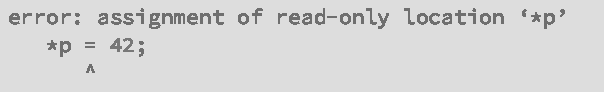




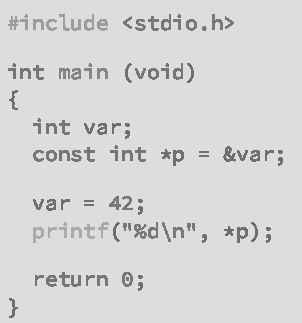
Entonces desde un función se puede modificar una variable declarada en el main u otra función (siempre que esta no haya finalizado) usando direcciones como argumentos y punteros como parámetros en las funciones llamadas.

CALIFICADOR CONST

Como se vio, usar el calificador const evita que se modifique una variable, intentar hacerlo es un error de compilación:

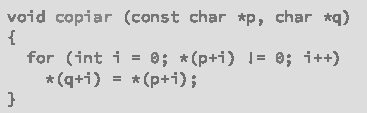


Sin embargo, la variable podría modificarse accediendola directamente por su nombre, ya que la variable no fue definida con const.





Esta característica se usa generalmente en funciones para asegurarse que los datos pasados no sean modificados.



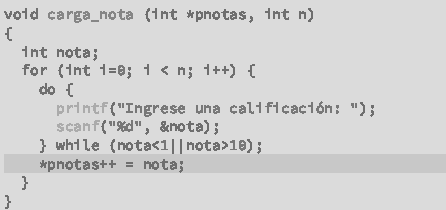
En esa función, la expresión const char \*p asegura que la cadena apuntada por p no

sea modificada en la función.

Hay cuatro formas de usar el calificador const en la lista de parámetros asociados a punteros:

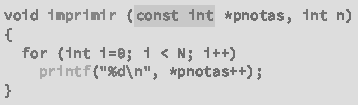
* PUNTERO NO CONSTANTE A DATO NO CONSTANTE

Se puede modificar el valor del puntero y el valor de la variable a la que apunta:



* PUNTERO NO CONSTANTE A DATO CONSTANTE

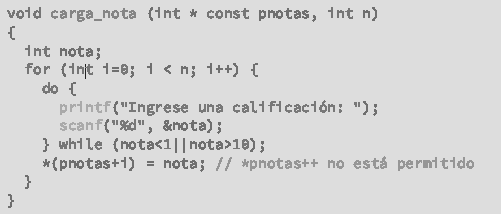
Se puede modificar el valor del puntero y pero no el valor de la variable a la que apunta:

El puntero apunta a un tipo const int.

El valor del puntero se puede modificar.

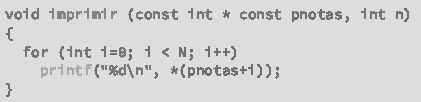
* PUNTERO CONSTANTE A DATO NO CONSTANTE

No se puede modificar el valor del puntero y pero si el valor de la variable a la que apunta.

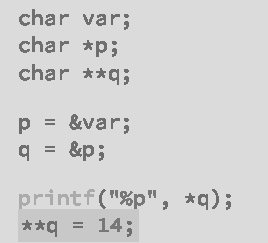


* PUNTERO CONSTANTE A DATO CONSTANTE

No se puede modificar el valor del puntero y no se puede modificar el valor de la variable a la que apunta.



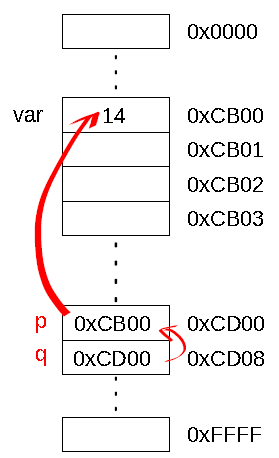
PUNTEROS A PUNTERO



Si p apunta a un tipo char, el puntero q así definido podrá apuntar a un tipo char \*.

Se dice que q es un puntero a puntero.

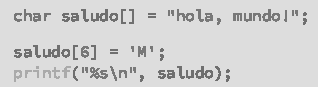
En esta línea (\*\*q = 14;), el operador de desreferencia que tiene mayor precedencia es el de la derecha, que opera sobre el puntero q

Equivale a encerrarlos entre paréntesis desreferenciar q (ej: \*(\*q)) , o sea (\*q), trae el contenido del puntero p (que es la dirección de var)

Entonces al hacer la doble desreferencia de q se accede a la variable var

**PUNTEROS EN LENGUAJE C (TERCERA PARTE)**

ARREGLOS DE CARACTERES (YA VISTO)

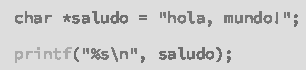


Para almacenar cadenas se vieron los arreglos de caracteres, su inicialización y su uso char

La cadena saludo así definida tiene el tamaño justo para la frase hola, mundo! + un carácter para el cero de final de línea

La cadena se puede modificar siempre y cuando no se supere el tamaño original

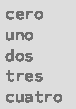
PUNTEROS A CADENAS CONSTANTES



Otra manera de almacenar cadenas es con los punteros a cadenas de caracteres

La diferencia fundamental es que no se pueden modificar. Se pueden considerar cadenas constantes

ARREGLOS DE CADENAS

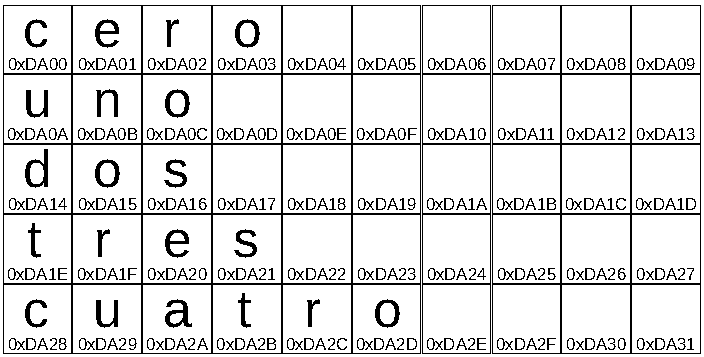


Para almacenar varias cadenas se puede tener un arreglo bidimensional de caracteres

Esto define un arreglo de 5 filas, donde el tamaño máximo de

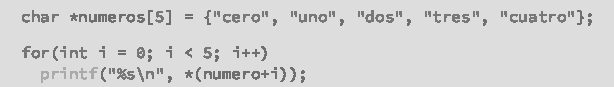
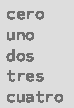
cada fila es 10 caracteres

La expresión numeros[i] devuelve la dirección de memoria del primer elemento de la fila i



Las casillas vacías en realidad contienen el número cero (no el caracter '0') pero se omiten para hacer más claro el gráfico

ARREGLOS DE PUNTEROS



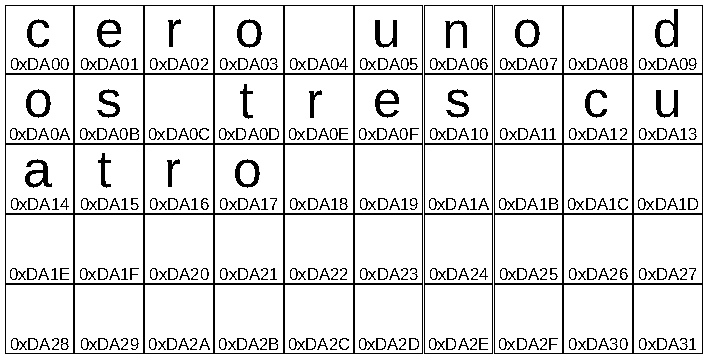
Para almacenar varias cadenas también se podría usar un arreglo de punteros a caracteres

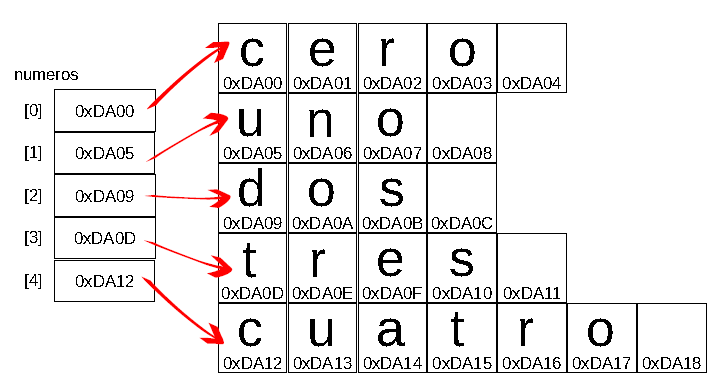
Esto es arreglo de 5 elementos de tipo puntero a char

Cada puntero apunta a una cadena de texto, que no puede ser modificada

ORGANIZACIÓN DE LOS DATOS CON ARREGLOS DE PUNTEROS

La organización de los datos cuando se utiliza un arreglo de punteros es diferente





PUNTEROS VS. ARREGLOS MULTIDIMENSIONALES

La principal diferencia entre los arreglos de punteros y los arreglos multidimensionales es que, volviendo al ejemplo de los números, es que en el caso de los arreglos multidimensionales las expresiones:

numeros

numeros[0]

&numeros[0]

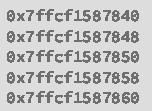
&numeros[0][0]

hacen referencia todas a la misma dirección

Sin embargo, en el caso de los arreglos de punteros esto no es así

En el caso de

el arreglo consta de 5 punteros, es decir, cada elemento tiene un tamaño de 8 bytes



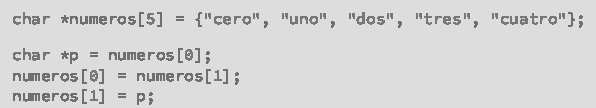
Además, como en todos los arreglos, la dirección del primer elemento es igual a la dirección devuelta por la expresión que corresponde solo al nombre del arreglo, entonces se puede decir que en cualquier caso numeros es igual a &numeros[0]

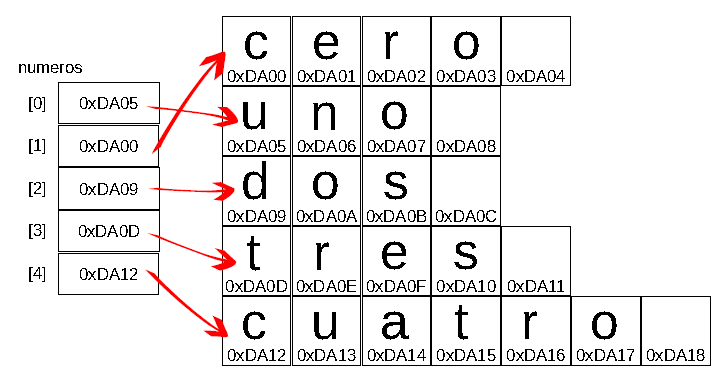
Si bien numeros[0] corresponde al primer elemento del arreglo, el valor contenido es la dirección de memoria donde se encuentra la cadena. Entonces la expresión numeros[0] es igual a &numeros[0][0]

Sin embargo los primeros dos son diferentes de los segundos 58 / 98

ORDENAMIENTO DE ESTRUCTURAS UTILIZANDO ARREGLO DE PUNTEROS

Se puede aprovechar el hecho de que los punteros se pueden intercambiar para intercambiar filas de un arreglo en lugar del intercambio elemento por elemento de un arreglo bidimensional





Las cadenas no cambian su valor ni posición, cambia el orden de sus apuntadores

INICIALIZACIÓN DE PUNTEROS Y RESERVA DE ESPACIO EN MEMORIA: MALLOC, CALLOC, REALLOC Y FREE



Así como la sentencia hace una reserva de un espacio de memoria de 4 bytes (en mi máquina los enteros tienen 4 bytes), y la sentenciadefine e inicializa un puntero apuntando a esa posición de

memoria...

...es posible hacer la reserva de memoria sin hacer la declaración de una variable usando la función malloc declarada en stdlib.h.



Se llama con la cantidad de memoria que se necesita reservar como argumento, expresada en bytes

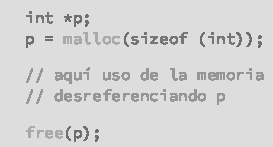
La función devuelve un puntero a la primera posición de la memoria reservada o un NULL si no se pudo hacer la reserva

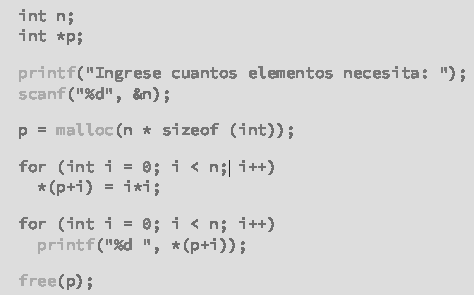
Para no cometer errores en el tamaño pedido o poder usar el programa en distintas arquitecturas, en lugar de pasar un valor literal de memoria solicitada se pasa la expresión sizeof (int) o del tipo necesario

o también usando el puntero desreferenciado int

Cada vez que se asigne memoria mediante la función malloc debe tenerse en cuenta que una vez finalizado el uso de la memoria, debe liberarse la misma con la función free

Debe ser llamada pasando como argumento el puntero devuelto en su momento por malloc



También puede reservarse memoria para un arreglo de tamaño n

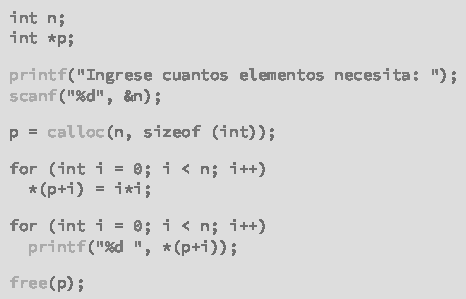
Esto es más eficiente que declarar un arreglo de tamaño sobredimensionado para el problema en cuestión y después solo atender a las necesidades del usuario

Esto también es preferible antes que usar los arreglos de dimensión variable los cuales no están admitidos en esta materia

La asignación de memoria realizada por malloc no modifica el contenido de esa memoria y hay que tenerlo en cuenta para usar estas direcciones sin inicializar.

En cambio, la función calloc también asigna memoria, pero inicializado en cero a los espacios de memoria reservados

La función calloc también devuelve el puntero a la primera posición de memoria reservada, pero necesita dos argumentos: la cantidad de elementos y, el tamaño de cada elemento



En ocasiones es necesario cambiar el tamaño del arreglo

La biblioteca estándar también cuenta con la función realloc la cual puede redimensionar el arreglo

Como las otras funciones de asignación, realloc devuelve el puntero a la posición de memoria (re)asignada

Debe ser llamada con el puntero de la memoria que se quiere modificar como primer argumento, y el nuevo tamaño como segundo argumento

Siempre es conveniente chequear que el valor del puntero devuelto por malloc, calloc o realloc no sea NULL lo que indicaría que no fue posible la asignación, de lo contrario podría ser un error en tiempo de ejecución

Liberar por segunda vez un puntero da un error en tiempo de ejecución

Desreferenciar un puntero que apunta a memoria ya liberada también es un error en tiempo de ejecución

PUNTEROS A FUNCIÓN

Los punteros también pueden apuntar a direcciones de memoria donde están implementadas las funciones

Los punteros a funciones pueden ser pasados a otras funciones o ser devueltos por funciones

Los punteros a funciones también pueden almacenarse en arreglos y asignarse a otros punteros a funciones

Al igual que con los punteros a variables, los punteros a funciones deben ser definidos antes de su uso

En las declaraciones de los punteros a funciones, debe aclararse que tipos de valores recibirá la función apuntada y que tipo de valor devolverá



Por ejemplo:

es la definición un puntero a una función que espera dos enteros y devuelve un entero

Es importante diferenciar el prototipo de una función común y corriente que devuelve un puntero a un entero

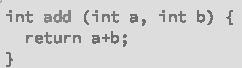


de un puntero a una función que devuelve un entero



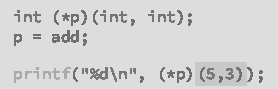
El uso de los paréntesis es la única forma de diferenciar entre los dos casos

Al igual que los arreglos, el nombre de la función indica la posición de memoria donde comienza su implementación

Entonces si se tiene una definición de función como

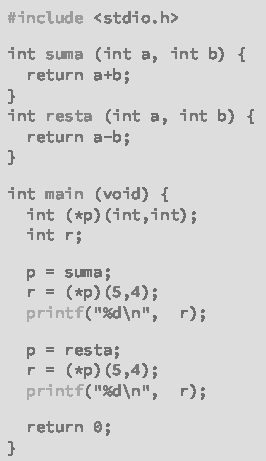
Se puede hacer una asignación como

El llamado a las funciones apuntadas por medio de la desreferencia se puede hacer con



En lugar de usar el nombre de la función se usa el puntero desreferenciado

El paso de argumentos se hace encerrando la lista entre paréntesis como en una función común



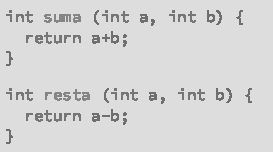
La expresión r = (\*p)(5,4); podría escribirse



sin usar el operador de desreferencia

Sin embargo esto no es aconsejable porque se pierde la información a cerca de p... Alguien podría pensar que es una función que no está definida

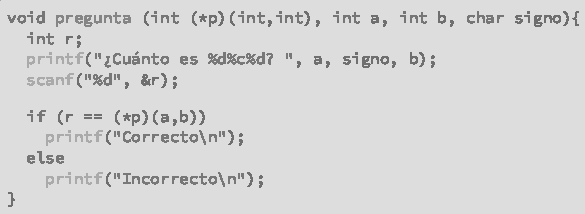
Se aconseja usar la versión con el operador de desreferencia



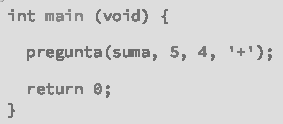
Un ejemplo un poco más complejo es el siguiente.

Supongamos las dos funciones ya vistas que resuelven una operación, ambas reciben dos enteros y devuelven un entero

En lugar de definir el puntero en la función main, se lo pone en la lista de parámetros de una función, que recibe además los operandos que se le pasan al puntero a función

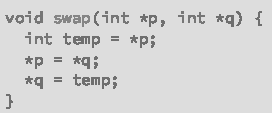


El llamado a la función, se hace simplemente pasando el nombre de la función



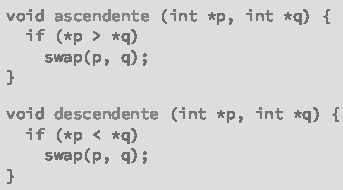
También se puede enviar otra función

* EJEMPLO DE ORDENAMIENTO

Se quiere ordenar un arreglo de enteros, con la posibilidad de que sea ascendente o descendente según se pida

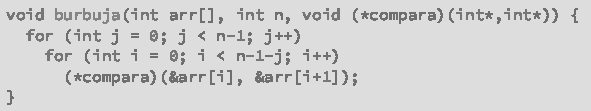
Primero, se define la función donde se van a intercambiar los valores (independientemente de que sea ascendente o descendente)

Luego las funciones para cada caso

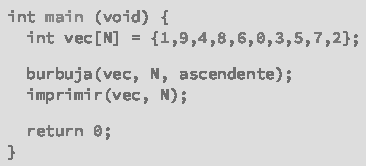


En las funciones solo cambia el signo del condicional. En los punteros p y q vendrán direcciones de dos elementos consecutivos del arreglo

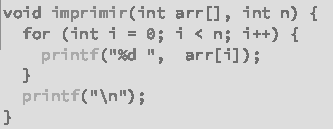
La función clave es donde se realiza el método de la burbuja, que recibe la dirección de la función elegida para hacer el ordenamiento, la cual es llamada pasando dos elementos consecutivos del arreglo



En el main simplemente se llama a la función burbuja pasándole el arreglo y la función que se elige para ordenar



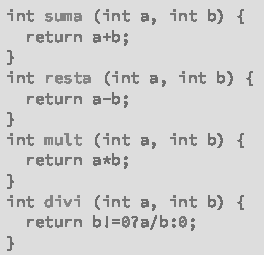
Se agrega la función ya usada para imprimir arreglos

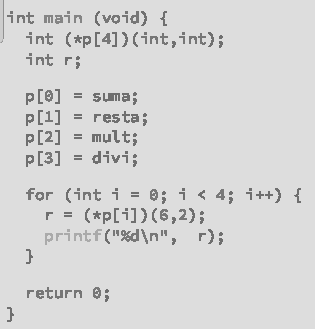


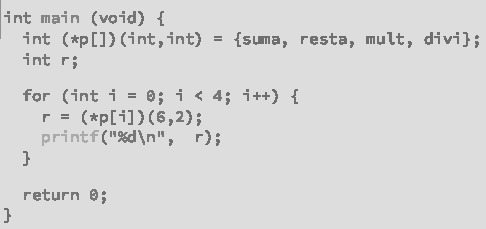
ARREGLO DE PUNTEROS A FUNCIÓN

Como con los demás tipos de variables, se puede hacer un arreglo de punteros a función

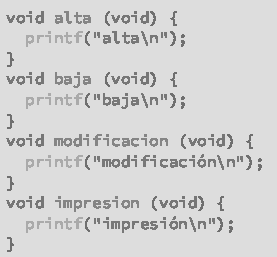
Por ejemplo, si se tienen 4 funciones definidas como



Entonces

O simplemente inicializando en la definición del arreglo

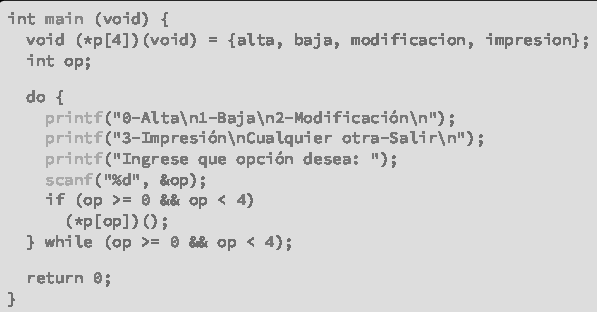
* EJEMPLO DE MENÚ CON ARREGLO DE PUNTEROS A FUNCIÓN



Se trata de reemplazar la estructura se selección múltiple switch por una estructura repetitiva con punteros a funciones

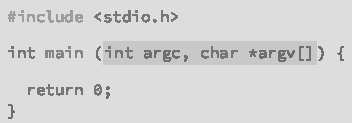
Para simplificar, en el ejemplo las opciones solo imprimen como se llama la función que la ejecuta

Las funciones son —---->



Dependiendo de la variable op se ejecutará la función que corresponda del arreglo

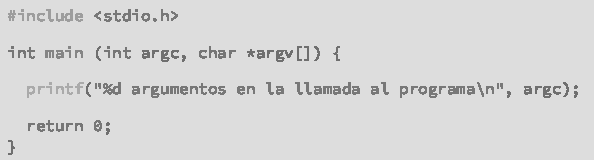
ARGUMENTOS POR LÍNEA DE COMANDOS



El estándar de C permite que los parámetros de main sean: >void o como en el siguiente ejemplo

El primer parámetro, argc, es igual a la cantidad de argumentos usados en la llamada al programa

Si se tiene el archivo argumentos-main-01.c con



Se lo compila con

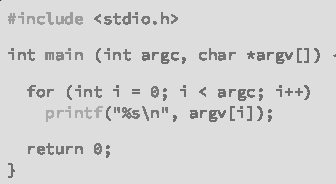


El ejecutable ahora se llama test



Si se lanza el programa con palabras extras como argumento su salida será

Se puede ver que test se ejecutó de manera correcta, y en argc se cargó un 4 correspondiente a los 3 argumentos pasados en el llamado al programa y el programa mismo



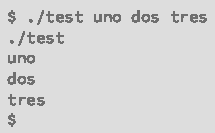
El parámetro char \*argv[] es un arreglo de punteros a cadenas y cada una es uno de los argumentos pasados a la llamada del programa

Si tenemos el archivo argumentos-main-02.c con —---->

Se puede imprimir cada uno de los parámetros, compilando con:



si se lanza este programa de la misma manera que la vez anterior…



* EJEMPLO PARA ABRIR UN ARCHIVO (SOLO LA PRIMERA PARTE)

