Apuntador:

variable que almacena una direccion de memoria

int \*ptr=&x; //Llevo a ptr la direccion de x

\*ptr=4; // Llevo a la direccion que contiene ptr 4 (es decir x=4)

ptr++; //aumento 1 segun el tipo, como es int aumenta 4

\*(ptr+1)=40; //No modifica ptr

\*(ptr++)=40; //Modifica ptr probablemente despues de hacer la operacion (llevar 40)

char \*a; //Cuando sume uno va a sumar 1

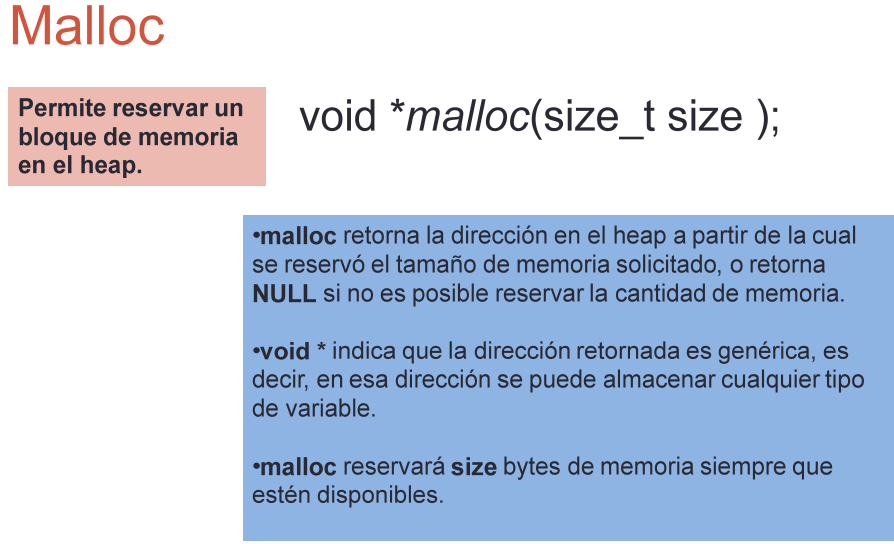
char (\*b)[10]; //Cuando sume uno va sumar 10 en realidad

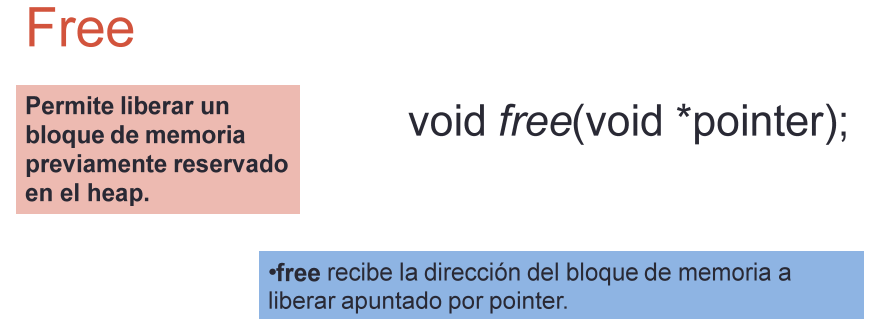
Para declararlo: int \*pvar1;

pvar1=&var1; //& obtiene la dirección de una variable

pvar1=\*var1; //\* obtiene el contenido de una variable

Memoria:

Para declarar variables en el heap se utilizan: malloc y free



Main con parametros:

<http://decsai.ugr.es/~jfv/ed1/c/cdrom/cap6/cap64.htm>

1. pvar2=&var2;
2. pvar1=&pvar2;
3. int \*py

py=&y py apunta a Y, que está en la dirección 8

\*py=\*px Ponemos a apuntar py a la dirección de px

4. px vale 8 (que apunta a y), si le restamos 1, px vale 7, que estaria contenido en la direccion 4(x), px=4(x), luego \*px=5 implica que a lo que hay en la direccion 4(x) queda valiendo 5, es decir x=5

5. a=2; b=4; c=-1; d=-2

**// instrucciones**

int a,b=3,c=8, d; /\* a=0, b=3, c=8, d=0 \*/

int \*p1 = &a, \*p2, \*p3 = &c; /\* p1 apunta a la direccion de a, p3 apunta a la direccion de c \*/

\*p1 = 2; /\* a lo que apunta p1 llevele 2, a=2 \*/

p2 = p3; /\* a p2 llevele p3 c \*/

\*p2 = \*p1-b; /\* a lo que apunta p2 llevele lo que apunta p1-b, c=2-3=-1 \*/

d = (\*p2)\*(\*p1); /\* a d llevele lo que apunta a p2 \* lo que apunta p1, d=(-1)\*(2)=-2\*/

p3 = &d; /\* p3 el contenido de d \*/

b = a + b + c; /\* b=2+3+(-1)=4 \*/

6. El **stack** permite almacenar argumentos y variables locales durante la ejecución de las funciones en las que están definidas.

7. Porque lo que se quiere es modificar el valor de la variable que se manda como parámetro y para esto se debe mandar la referencia a ella y no su valor.

swap (&x,\*p)

8. A la funcion f1 se le estan enviando como parametros los enteros d3 y d1 y deberia enviarsele dos apuntadores.

En la funcion f1 z no es visible dentro de esta función por lo que la instruccion \*y=x+z; hace uso de un z que no se tiene.

La salida es :

**d1=3, d2=1, d3=4**

**d1=2, d2=0, d3=2**

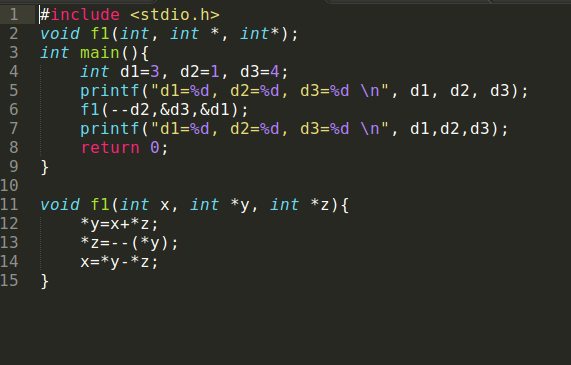
d2 cambia porque antes de enviarse como parámetro por valor, se decrementa en 1.

ya dentro de f1, las variables se podrían mirar así:

\*y = 0 + \*z; /\* d3 = (x=0)+(d1=3) = 3 \*/

\*z = --(\*y); /\* decrementa d3, --d3 = 2, y luego, d1 = 2 \*/

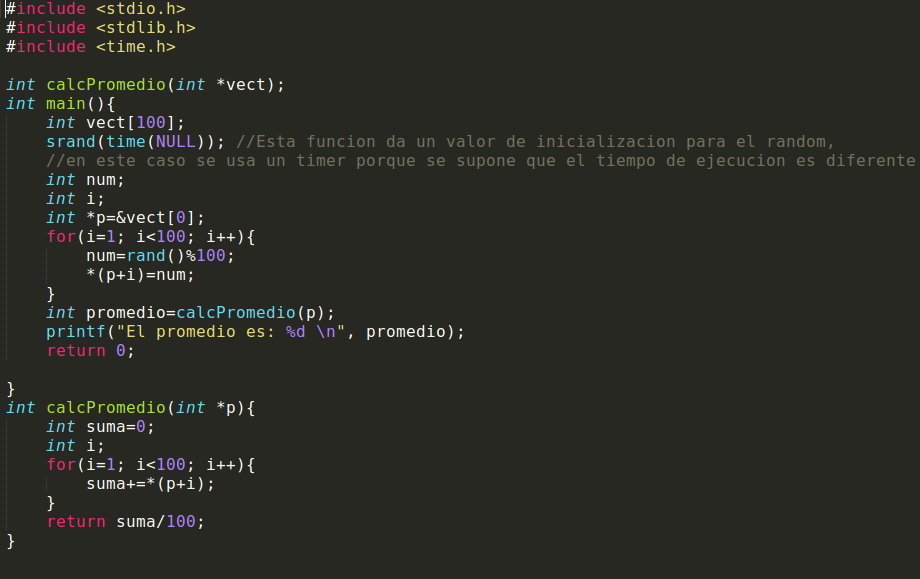
x = \*y - \*z; /\* x = d3 - d1 = 3 - 2 = 1 \*/

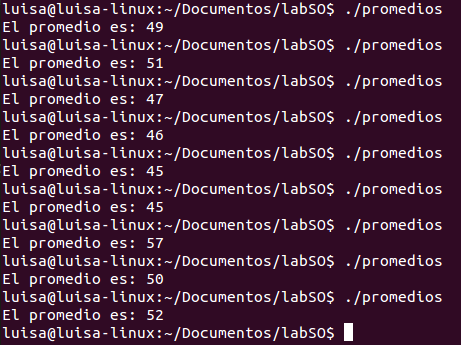


9. vect= vect[0]--> vect+1=vect[1] a=20 y b=2

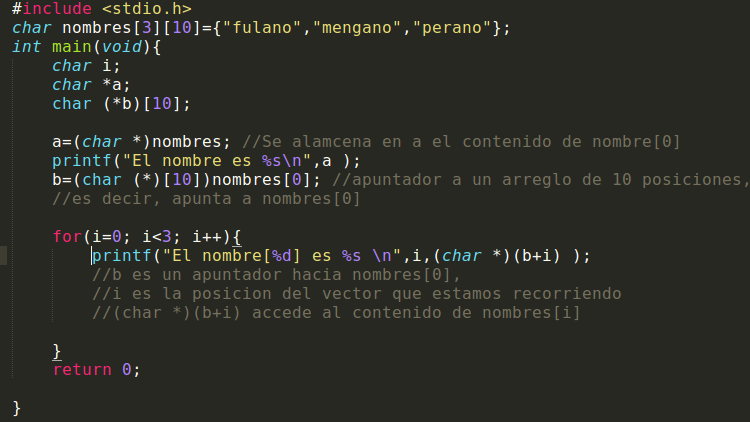
10. A: [ 2, 3, **5**, 0, **11**, 6 ]

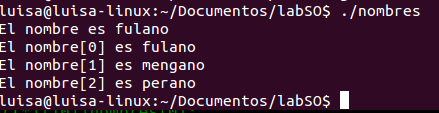
11.





12.



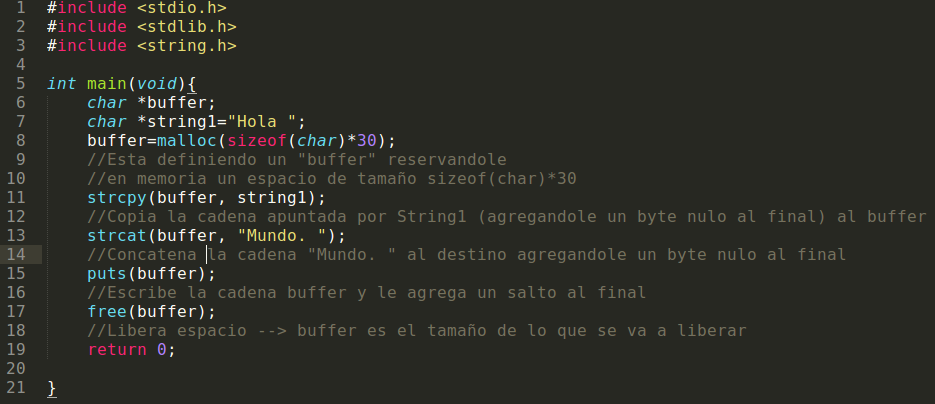


13. Para reservar memoria se debe saber exactamente el número de bytes que ocupa cualquier estructura de datos. La función **sizeof()**, devuelve el tamaño en bytes necesaria para almacenar un objeto del tipo del operando.

sizeof(char) : averigua cuantos bytes ocupa un char

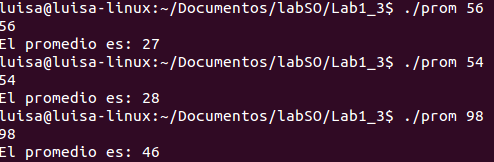
malloc(sizeof(char)\*30): el # de bytes que ocupa un char se multiplica por 30.

14.





15. 



16.

