Explicación códigos de C

Ejercicio 1:
Se inicializa una variable X como un entero, luego, se le asigna el valor de una expresión matemática, la cual seguidamente se imprime, dando como resultado el valor entero resultante de la expresión dada, este proceso se realiza 4 veces, ya que luego de imprimir el valor de la primera expresión, a la variable se le asigna como valor otra expresión diferente y se imprime su resultado, proceso que se realiza 4 veces dando los resultados de 11, 1, 0, 1 respectivamente al correr toda la función.

Ejercicio 2:
Se definen 3 variables (x, y, z) de las cuales solamente a ‘x’ se le asigna un valor, siendo este el 2.
En la linea siguiente se realiza una sencilla operación de multiplicación, donde se multiplica al valor asignado a ‘x’ (el 2) el resultado de la suma 3+2 (5) por lo que da como resultado el 10, luego, pasamos a una línea donde debido al orden de asignación del lenguaje, se le asigna el valor de 4 a la variable ‘z’, luego se iguala el valor de ‘y’ a lo que tenga la variable ‘z’ (en este caso el 4) y luego se realiza la misma multiplicación de la línea anterior, ‘x’ valiendo 10 (por la multiplicación ejecutada en la línea anterior) multiplicado por ‘y’ (4) dando un resultado de 40.
En la línea siguiente se realiza una comparación entre los valores de las variables ‘y’ y ‘z’, ambas en este momento guardan el valor 4, por lo que a esa comparativa se le asigna el valor de 1 (ya que esta es la forma de mostrar una comparación booleana en entero según el lenguaje, 1 sería True en este caso) y luego se iguala el valor de ‘x’ al de la comparación, ‘x’ termina con el valor de 1.
Por último, se igualan los valores de ‘y’ y ‘z’, ambos seguían guardando el valor 4, por lo que la igualación no afecta en nada, luego se realiza un comparación entre la variable ‘x’ y el resultado de la igualación, o sea, se compara que ‘x’ (que en este momento vale 1) sea igual al valor que guardan ‘y’ y ‘z’ (en este caso el 4), la comparación da como resultado que no son iguales, dicho de otra forma un False, pero la función ‘PRINTX’ que imprime el resultado en consola y se llama en cada línea de la función, imprime lo que guarda la variable ‘x’, por lo que la comparación no hace nada y se imprime de nuevo el valor de 1 que es el valor asignado a ‘x’ desde la línea de código anterior.

Ejercicio 3:
Se inicializan las variables ‘x’, ‘y’ y ‘z’ como enteros, seguidamente se le asignan los valores 2 a ‘x’, 1 a ‘y’ y 0 ‘z’,, posteriormente se hace una igualación donde se utiliza un OR lógico (||) y un And lógico (&&), que se traduce en que la operación AND se realiza primero, al ser ambos valores en ‘x’ y ‘y’ enteros positivos, el resultado daría True (1), seguidamente se hace el OR, el cual al tener el resultado anterior como una de sus opciones devuelve True (1), ese valor de 1 se le asigna a la variable ‘x’, la cual se imprime, dando el primer resultado (int = 1).
La siguiente línea se igualan primero los valores de ‘x’ y ‘y’ a 1, seguidamente, se realiza un incremento (++) a la variable ‘x’ (lo que convierte su valor a 2), sin embargo, la operación realizada sería 1-1, lo que dá como resultado 0, valor que se le asigna a ‘z’, con esto hecho se imprime el valor de ‘x’ que es 2 y el valor de ‘z’ que es 0.
Seguidamente hay una expresión que se evalúa en partes, primero, se realiza otro incremento a ‘x’, el cual pasa a tener el valor de 3, la operación ++y, pre-incrementa el valor de ‘y’ (que es 1) a 2 y utiliza ese valor, dando como resultado una ecuación que se vería como un -2+2 (debido a que el incremento de ‘x’ no se utiliza aún), y dá como resultado 0, valor que se le asigna a la variable ‘z’, para finalizar de imprimen los valores de ‘x’ y ‘z’, siendo estos respectivamente 3 y 0.
En la última línea, se pre-incrementa el valor de ‘x’ en la ecuación, dejándola con el valor 4, ese valor se utiliza para hacer una división que se vería como 4/4, lo que da como resultado 1, valor que se le asigna a la variable ‘z’ y que luego se imprime.

Ejercicio 4:
El código inicializa 3 variables, ‘x’, ‘y’ y ‘z’, las define como enteras y seguidamente se le asignan los valores de 03 a ‘x’, 02 a y´ y 01 a ‘z’, seguidamente se realizan operaciones de bits a los valores de las variables, en la primer línea, se realiza primero un ‘y & z’, el ‘&’ crea un mapa de bits nuevos si en la misma posición de los valores en binario hay un 1, lo coloca o si no coloca 0, dicho de otra forma, ‘y’ en binario tendría el valor 10 (2 en decimal) y z el valor 01 (1 en decimal), sin embargo, ninguno de los 2 valores comparte un 1 en la misma posición, lo que da como resultado 0, este 0 se evalúa con una operación ‘|’ la cual si en alguno de sus valores hay un 1, esta retorna 1 como resultado, en este caso, se realiza 11 | 0 (11 siendo 3 en decimal y valor guardado en ‘x’ y 0 el valor obtenido anteriormente), esto da como resultado el 11, en cual se imprime como 3 en consola.
En la siguiente línea (y para las demás antes de que se diga lo contrario) los valores de ‘x’, ‘y’ y ‘z’ no cambian, debido a que solo se llamaron y evaluaron en una función PRINT, por lo que en esta línea se realiza primero una sustitución en ‘-z’ ya que sería -1, seguidamente se evalúa ‘y & -z’, que sería 2 & -1, -1 en binario es una representación de complemento a dos con todos los bits en 1 y el 2 se vería como un 10, debido a esto, la operación AND, da como resultado 10 que representa al 2, con esto hecho, se evalúa el ‘x | 2’, que sería 3 | 2, por las explicaciones anteriores se entiende como esto da como resultado 11, que sería 3, valor que se imprime en consola.
Seguidamente viene la operación ‘(x ^ y & - z)’, primero se evalúa el ‘y & -z’, que es 2 (igual a lo visto en la línea anterior), luego el ‘x ^ 2, que se vería como 3 ^ 2, el símbolo ^ es un XOR, que en un mapa de bits, si donde hay un 1 en uno de los valores, y en la misma posición del otro valor hay un 0, se retorna 1 y viceversa, retorna 0 si en la misma posición ambos valores tienen un 1 o un 0, con esto en cuenta 3 en binario es 11 y 2 es 10, por lo que la expresión retorna 01, que equivale a 1 en decimal, valor que se imprime en consola.
Luego viene la expresión ‘(x & y && z)’, primero se evalúa el ‘y && z’, que sería 2 && 1, ambos valores verdaderos por lo que el resultado es 1, luego el ‘x & 1’, que sería 3 & 1, 3 siendo 11 y 1 siendo 01, da como resultado 01, que es 1 valor que se imprime en consola.
Ahora sí se le reasignan nuevos valores a las variables ‘x’ y ‘y’, a ‘x’ se le asigna un 1 y a ‘y’ un -1, luego de esto, se realiza la operación ‘(! x | x)’, primero se evalúa el ‘! x’, que se vería como !1, una negación lógica, que da como resultado 0, luego se realiza el ‘0 | x’ que se vería como 0 | 1, lo que da 1 como resultado, valor que se imprime en consola.
Luego viene la operación ‘(- x | x)’, se evalúa el -1 que en binario son todos 1s y el 1 que se vería como 01 en la operación OR, esto da todos 1s, por lo que el resultado es -1, valor que se imprime en consola.
Luego, viene la operación ‘(x ^ x)’, se evalúa 1 ^ 1, por ser XOR y ser ambos iguales, da como resultado 0, valor que se imprime en consola.
Luego se realiza una operación que se ve de la forma ‘(x «= 3)’, esto quiere decir que ‘x’ se desplaza 3 bits a la izquierda, dicho de otra forma, si 1 en binario es 0001, desplazado 3 posiciones a la izquierda sería 1000, que es equivalente al 8 en decimal, valor que se imprime en consola.
Luego se realiza lo mismo con el valor en la variable ‘y’ ‘(y «= 3)’, -1 en binario son todos 1, por lo que desplazado 3 veces a la izquierda, por la aritmética del complemento a dos, da como resultado -8, valor que se imprime en consola.
Por último, se tiene la operación ‘(y »= 3)’, en este caso ‘y’ se desplaza a la derecha en 3 bits, como en este momento y vale -8, es hacer a la inversa lo realizado en la operación anterior, por lo que el resultado sería -1, valor que se imprime en consola.

Ejercicio 5:
Primero se inicializan las variables enteras ‘x’, ‘y’ y ‘z’, y se les asigna el valor 1 a cada una.
Se realiza una suma que se ve como ‘x += y += z’ y se traduce en que primero al valor de la variable ‘y’ se le suma el valor de ‘z’, o sea 1+1, esto da 2, seguidamente al valor de la variable ‘x’, se le suma este 2, dando como resultado 3, por lo que luego de esto, ‘x’ vale 3, ‘y’ vale 2 y ‘z’ sigue valiendo 1.

Ahora se llama a imprimir la operación ‘(x < y ? y : x)’, la condición ‘x < y’ o visto de otra forma 3<2 es falsa, por lo que el operador ternario evalúa ‘x’, dando como resultado el 3, valor que se imprime en consola.
Seguidamente se realiza una operación que se ve de la forma ‘(x < y ? x ++ : y ++)’, primero se compara ‘x < y’, que se vería como 3 < 2, lo cual es falso, por ende, se realiza la acción de incrementar y con ‘y++’, sin embargo, se imprime ‘y’ antes del incremento, por lo que en consola aparece impreso un 2, luego llama a imprimir lo que haya en ‘x’ y luego en ‘y’, por lo que seguidamente se imprimen un 3 (Valor en ‘x’) y otro 3 (valor en ‘y’ después del incremento).
Luego viene la operación ‘(z += x < y ? x ++ : y ++)’ donde se lee primero ‘x < y’. O sea 3 < 3, lo cual es falso, por ende se evalúa el ‘y ++’, sin embargo primero se imprime el valor de ‘y’ antes del incremento, el cual es 3, ‘z’ incrementa por 3, lo que lo deja valiendo 4 al igual que ‘y’, para terminar imprimiendo en consola el valor de ambos, primero ‘y’ y luego ‘z’, mostrando 2 4 seguidos en consola.
Luego se le asigna a la variable ‘x’ el valor de 3 y a ‘y’ y ‘z’ el valor de 4.
Se realiza la operación ‘((z >= y >= x) ? 1 : 0)’, ‘z >= y’ se evalúa primero, visto de otra forma 4 >= 4, lo cual es verdadero, retorna 1, luego se evalúa ‘1 >= x’, visto de otra forma 1 >= 3, que es falso, retorna 0, por lo que se evalúa el operador ternario 0, valor que se imprime en consola.
Por último, se evalúa la expresión ‘(z >= y && y >= x)’, primero se evalúa ‘z >= y’, que ya se vio que es 1, y luego se evalúa ‘y >= x’, esto se puede ver como 4 >= 3, que es verdadero, retorna 1, al ser ambas condiciones verdaderas, la expresión retorna como resultado un 1, valor que se imprime en consola.

Ejercicio 6:
Se inicializan las 3 variables enteras ‘x’, ‘y’ y ‘z’, para seguidamente igualar las 3 al valor 1.
Se evalúa primero ‘++x’, incrementando ‘x’ a 2, ‘++y && ++z’ no se evalúan ya que ‘++x’ es verdadero, por lo que ‘x’ y ‘z’ no cambian, por lo que se imprime en consola ‘x=2 y=1 z=1’
Se vuelvan a inicializar las 3 variables con el valor 1.
Se evalúa la operación ‘++x && ++y || ++z’ y de aquí se evalúa primero ‘++x’, incrementando ‘x’ a 2, es un valor verdadero, por lo que se procede a evaluar ‘++y’, esto convierte a ‘y’ en 2, el cual también es un valor verdadero, al ser la operación ‘++x && ++y’ la operación ‘++z’ no se evalúa, por lo que la salida en consola sería ‘x=2 y=2 z=1’.
Se vuelven a inicializar todas las variables a 1.
Se evalúa la operación ‘++x && ++y && ++z’ , comenzando por ‘++x’ que incrementa ‘x’ a 2, valor verdadero, por lo que pasa a evaluar ‘++y’, incrementando ‘y’ a 2, valor también verdadero por lo que procede a evaluar ‘++z’, incrementando ‘z’ a 2, valor igualmente verdadero, por lo que la salida en consola se vería de la forma ‘x=2 y=2 z=2’.
Se vuelven a inicializar las variables, pero ahora con el valor -1.
Se evalúa la operación ‘++x && ++y || ++z’, comenzando por ‘++x’, esto incrementa ‘x’ a 0, valor que se toma como falso, por lo que la evaluación del ‘&&’ no evalúa a ‘++y’, debido a que el operador ‘&&’ es falso, se procede a evaluar el ‘++z’, incrementando su valor a 0, valor que se toma como falso, sin embargo, la salida retorna en consola los valores de las variables, por lo que la salida se vería de la forma ‘x=0 y=-1 z=0’.
Se vuelven a inicializar las variables a -1
Se evalúa la operación ‘++x || ++y && ++z’, comenzando por la operación ‘++x’, lo que incrementa x al valor 0, lo cual es tomado como falso, por lo que se procede a evaluar la operación ‘++y && ++z’, se evalúa primero ‘++y’ incrementando la variable a 0, que es falso, por lo que ‘++z’ no se evalúa, dando como resultado en consola que ‘x=0 y=0 z=-1’.
Se vuelven a inicializar las variables con el valor -1.

Se evalúa la operación ‘++x && ++y && ++z’, comenzando por ‘++x’, que incrementa el valor de ‘x’ a 0, puesto que esto es un valor que retorna que es falso, el resto de la operación no se evalúa, dando como resultado en consola que ‘x=0 y=-1 z=-1’.

Ejercicio 7:
Se recibe como entrada de la función una cadena definida como input7 que se ve de la siguiente manera ‘char input7[] = "SSSWILTECH1\1\11W\1WALLMP1"’.
Dentro de la función, primero se inicializan las variables enteras ‘i’ y ‘c’.
El código realiza un bucle for, que comienza desde el tercer carácter de la cadena ‘input7’ y procesa cada carácter hasta el final de la línea.
La función switch (c) que se encuentra dentro del ciclo for evalúa el valor de ‘c’ y ejecuta una acción según el que sea, los casos serían:

- Caso ‘a’:

Si ‘c’ es ‘a’, imprime ‘i’ y continúa con el siguiente valor.

- Caso ‘1’:

Si ‘c’ es ‘1’, imprime ‘i’ y continúa con el siguiente valor.

- Caso 1:

Si ‘c’ es el carácter especial ‘\1’, entra a un ciclo while que avanza ‘i’ espacios hasta encontrar otro ‘\1’ o el final de la cadena.

- Caso 9:

Si ‘c’ es el carácter especial ‘\11’, imprime ‘s’.

- Caso ‘E’ o caso ‘L’:

Si c´ es ‘E’ o ‘L’, continua con el siguiente valor.

- Default:

Si ‘c’ no coincide con ninguno de los valores anteriores, imprime el carácter tal cual.
Si ‘switch’ no encuentra un ‘continue’ o ‘break’, ejecuta ‘putchar(‘ ‘)’ al final de la iteración, al terminar el ciclo, se ejecuta ‘putchar('\n')’, que agrega un salto de línea.
El resultado de la transformación de la cadena ‘input7’ luego de entrar al ciclo for se ve en consola como el mensaje ‘SWITCH SWAMP’.

Ejercicio 8:
Se crea el arreglo ‘a8’ de la siguiente forma ‘int a8[] = {0,1,2,3,4}’.
Se inicializan una variable entera ‘i’ y un puntero a entero ‘*p’
Luego se recorren una serie de ciclos for:

- for(i=0; i <=4; i++) PR(d,a8[i]):

Este primer ciclo recorre el arreglo ‘a8’ y usando índices imprime cada elemento, imprime en consola lo siguiente:
‘a8[i] = 0 a8[i] = 1 a8[i] = 2 a8[i] = 3 a8[i] = 4’

- for(p= &a8[0]; p<=&a8[4]; p++) :

El segundo ciclo utiliza punteros para recorrer el arreglo y acceder a los elementos, de esta forma imprime este resultado en consola:
‘*p = 0 *p = 1 *p = 2 *p = 3 *p = 4’

- for (p=&a8[0],i=1; i <=5; i++):

Este ciclo usa un puntero ‘p’ y accede a los elementos del arreglo desplazándose desde ‘p’, imprimiendo en consola lo siguiente:
‘p[i] = 1 p[i] = 2 p[i] = 3 p[i] = 4 p[i] = 0’.

- for(p=a8,i=0; p+1<=a8+4; p++,i++):

Este ciclo combina el uso de punteros y un índice ‘i’ para ejecutar la acción ‘PR(d,*(p+i))’, que imprime los valores en posiciones pares, imprimiento en consola el resultado:
‘*(p+i) = 0 *(p+i) = 2 *(p+i) = 4 *(p+i) = 0’.

- for (p=a8+4; p>=a8; p--) PR(d,*p):

Este ciclo recorre el arreglo hacia atrás utilizando punteros, imprimiendo el siguiente resultado:
‘*p = 4 *p = 3 *p = 2 *p = 1 *p = 0’.

- for (p=a8+4,i=0; i<=4; i++) PR(d,p[-i]):

Este ciclo usa un puntero ´p´ y accede a los elementos desplazándose negativamente desde ‘p’, imprimiendo el siguiente resultado:
‘p[-i] = 4 p[-i] = 3 p[-i] = 2 p[-i] = 1 p[-i] = 0’.

- for (p=a8+4; p >=a8; p--) PR(d,a8[p-a8]):

Este último ciclo recorre el arreglo hacia atrás usando punteros pero accediendo a los elementos mediante índices, imprimiendo en consola el siguiente resultado:
‘a8[p-a8] = 4 a8[p-a8] = 3 a8[p-a8] = 2 a8[p-a8] = 1 a8[p-a8] = 0’.

Ejercicio 9:
Se crea el arreglo de enteros ‘a9[]’ (‘int a9[] = {0, 1, 2, 3, 4}’), luego se crea el arreglo de punteros a enteros ‘*p9[]’, donde cada puntero apunta a un elemento de ‘a9’ (‘int *p9[] = {a9, a9 + 1, a9 + 2, a9 + 3, a9 + 4}’), por último ‘**pp9’ es un puntero a un puntero a entero, inicialmente apuntando al primer elemento de ‘p9’ (‘int **pp9 = p9’).

Luego se realizan una serie de macros, estos serían:

```
PR2(d, a9, *a9);
PR3(d, p9, *p9, **p9);
PR3(d, pp9, *pp9, **pp9);
NL;
```

Que imprimen:

- ‘a9’: la dirección del primer elemento de ‘a9’.
- ‘*a9’: el valor del primer elemento de ‘a9 (0)’.
- ‘p9’: la dirección del primer elemento de ‘p9’.
- ‘*p9’: la dirección del primer elemento de ‘a9 (igual que a9)’.
- ‘**p9’: el valor apuntado por el primer elemento de ‘p9 (0)’.
- ‘pp9’: la dirección de ‘p9’.
- ‘*pp9’: la dirección apuntada por ‘pp9 (el primer elemento de p9)’.
- ‘**pp9’: el valor apuntado por el primer elemento de ‘p9 (0)’.

Dando como resultado la siguiente salida en consola:
a9 = -1805840288 *a9 = 0
p9 = -1805840192 *p9 = -1805840288 **p9 = 0
pp9 = -1805840192 *pp9 = -1805840288 **pp9 = 0

Luego se realizan una serie de operaciones con ‘pp9’:

- pp9++; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘pp9++’ incrementa ‘pp9’ para apuntar al siguiente elemento.
‘PR3(d, pp9 - p9, *pp9 - a9, **pp9)’ imprime el índice actual de ‘pp9’ respecto a ‘p9’, el índice del puntero apuntado en ‘a9’ y ‘**pp9’ el valor apuntado, imprimiendo la salida:

‘pp9-p9 = 1 *pp9-a9 = 1 **pp9 = 1’

- *pp9++; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘*pp9++’ incrementa ‘pp9’ para apuntar al siguiente elemento en ‘p9’, lo demás es igual a la función anterior, esto imprime el siguiente resultado:
‘pp9-p9 = 2 *pp9-a9 = 2 **pp9 = 2’

- *++pp9; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘*++pp9’ incrementa ‘pp9’ antes de desreferenciarlo, lo demás es igual a las funciones anteriores, esto imprime el siguiente resultado:
‘pp9-p9 = 3 *pp9-a9 = 3 **pp9 = 3’

- ++*pp9; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘++*pp9’ incrementa el puntero al que ‘pp9’ está apuntando, moviéndolo al siguiente entero en ‘a9’, lo demás es igual a las funciones anteriores, esto imprime el siguiente resultado:
‘pp9-p9 = 3 *pp9-a9 = 4 **pp9 = 4’
Se resetea ‘pp9’ para que apunte al inicio de ‘p9’ (‘pp9 = p9’).
Se realizan otra serie de operaciones:

- **pp9++; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘**pp9++’ primero desreferencia ‘pp9’ para obtener ‘p9[0]’, incrementa ‘p9[0]’ (o sea, ahora es ‘p9[1]’) y luego incrementa ‘pp9’, por lo que este ahora apunta a ‘p9[1]’, lo demás es igual a las funciones anteriores, esto imprime en consola el siguiente resultado:
‘pp9-p9 = 1 *pp9-a9 = 1 **pp9 = 1’

- *++*pp9; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘*++pp9’ incrementa el puntero al que ‘pp9’ está apuntando, moviéndolo al siguiente entero en ‘a9’, lo demás es igual a las funciones anteriores, esto imprime el siguiente resultado:
pp9-p9 = 1 *pp9-a9 = 2 **pp9 = 2

- ++**pp9; PR3(d, pp9 - p9, *pp9 - a9, **pp9):

‘++**pp9’ incrementa el valor apuntado por ‘*pp9’, lo demás es igual a las funciones anteriores, esto imprime en consola el siguiente resultado:
pp9-p9 = 1 *pp9-a9 = 2 **pp9 = 3

Pregunta 10:

Primero se inicializa una matriz de 3X3 llamada ‘a10’ (‘int a10[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };’).
Luego se crea un arreglo de punteros llamado ‘pa’ (‘int *pa[3] = { a10[0], a10[1], a10[2] };’).
Por último un puntero ‘p10’(‘int *p10 = a10[0];’).
La función realiza 2 bucles for, el primero de estos, se ve de la forma:
‘for (i = 0; i < 3; i++)
PR3(d, a10[i][2-i], *a10[i], *(*(a10+i)+i));’
este itera sobre los índices o a 2 e imprime 3 valores en cada iteración:

1. ‘a10[i][2-i]’:

Accede al elemento en la posición ‘[i][2-i]’ de ‘a10’.

2. ‘*a10[i]’:

Desreferencia el puntero ‘a10[i]’, que apunta al primer elemento de la fila ‘i’.

3. ‘*(*(a10+i)+i)’:

Accede al elemento en la posición ‘[i][i]’ de ‘a10’.
Esto al terminar con cada índice va a imprimir lo siguiente en consola:
‘a10[i][2-i] = 3 *a10[i] = 1 *(*(a10+i)+i) = 1
a10[i][2-i] = 5 *a10[i] = 4 *(*(a10+i)+i) = 5

a10[i][2-i] = 7 *a10[i] = 7 *(*(a10+i)+i) = 9’

Seguidamente el segundo ciclo for también itera sobre los índices 0 a 2, pero esta vez imprime 2 valores:

1. ‘*pa[i]’:

Desreferencia el puntero ‘pa[i]’, que apunta al primer elemento de la fila ‘i’ en ‘a10’.

2. ‘p10[i]’:

Accede al elemento en la posición ‘i’ de ‘p10’, que es linealmente el mismo que ‘a10[0][i]’.
Esto al terminar con cada índice va a imprimir lo siguiente en consola:
‘*pa[i] = 1 p10[i] = 1
*pa[i] = 4 p10[i] = 2
*pa[i] = 7 p10[i] = 3’

Ejercicio 11:

Se crea un arreglo de punteros ‘*c’ (‘char *c[] = { "ENTER", "NEW", "POINT", "FIRST" };’).
Se crea otro arreglo de punteros ‘**cp’ (‘char **cp[] = { c+3, c+2, c+1, c };’).
Por último se crea un puntero a puntero ‘***cpp’(‘char ***cpp = cp;’).
En la primer línea de la función se incremente ‘cpp’ y desreferencia 2 veces, por lo que imprime ‘”NEW”’.
En la segunda línea ‘*++cpp’ incrementa ‘cpp’ para que apunte a ‘cp[2]’, luego desreferencia 1 vez, obteniendo ‘c+1’, luego ‘--*++cpp’ decrementa el puntero apuntado por ‘cpp’, por lo que ahora apunta a ‘c+1’, lo que desreferencia para obtener ‘”NEW”’, por último, ‘*--*++cpp+3’ avanza 3 caracteres desde ‘”NEW”’, imprimiendo ‘”R”’.
En la tercer línea, primero ‘cpp[-2]’ accede a ‘cp[0], que es ‘c+3’, luego ‘*cpp[-2]+3’ avanza 3 caracteres desde ‘”FIRST”’, imprimiendo ‘”ST”’.
Para finalizar en la cuarta línea ‘cpp[-1]’ accede a ‘cp[3]’, que es ‘c’, luego ‘cpp[-1][-1]’ accede al último elemento de ‘c’, que es ‘”FIRST”’, por último ‘cpp[-1][-1]+1’ avanza un carácter desde ‘”FIRST”’, imprimiendo ‘”IRST”’, dando como resultado que se imprima ‘”NEW POINT FIRST”’, sin embargo, por alguna razón se imprime en consola ‘”POINTER STEW”’.