3ra lista de fundamentos de programación (abstracta).

0. Al inicio, ‘u’ apunta a la dirección de memoria de ‘a’ #1256, y *u se le asigna **90**, por lo que ‘a’ ahora es **90**; ‘v’ tiene la dirección de memoria de ‘e’ #B837, entonces *v es **11**, pero ‘e’ aumenta en uno, entonces *v es **12**; ‘x’ obtiene el contenido de ‘v’ #E842, el cual es la dirección de ‘e’ #B837, por lo tanto, *x es **12**.

   *u tiene un nuevo valor, el cual es su resta en una unidad, es decir, **89**; *x es la suma de *u y **23**, y resulta **112**, esto implica que *v & ‘e’ cambien de valor; ‘y’ es la dirección de ‘c’ #06F2, ‘w’ obtiene la dirección de ‘b’ #A598; ‘d’ es el producto de ‘d’ y ‘e’, esto es **784**. ‘f’ se resta en uno, dando **12**; ‘z’ tiene la dirección de ‘d’ #9CE0 y *z es **784** (nuevo valor de ‘d’).

   ‘u’ obtiene la dirección que apunta ‘v’ #E842, siendo esta la dirección de ‘e’ #B837, por lo que *u es **112**; *z se multiplica por **6**, dando como nuevo valor **4704** y luego z obtiene la dirección de ‘e’ #B837 mediante la referencia de u y v; *w es la suma de a, b, c, d, e y f, siendo esto 89+3+5+4704+112+12 = **4925**, esto implica que ‘b’ tendrá este mismo valor; *w se resta con 2019, el resultado da **2906**, esto significa que ‘b’ cambia; ‘c’ aumenta en 1, dando **6**; ‘e’ disminuye en 1, obteniendo **111** y ahora *u y *v tienen este número.

   Ahora, ‘u’ vuelve a tener la dirección de ‘a’ #1256; ‘v’ obtiene la de ‘b’ #A598; & ‘w’ la de ‘c’ #06F2; *x es la suma de ‘d’ y *u, esto nos da 4704+111=**4793**, esta suma provoca que ‘e’ tengan este valor; *y es la suma de ‘e’ y *v, siendo esto 4793+2906=**7699**, esto implica que ‘c’ & *w cambie de valor; finalmente, *z es la suma de ‘f’ y *w, teniendo 12+7699= **7711** y ‘e’ cambia de valor a este último**.**

1. Se crean dos apuntadores de tipo arreglo de contenido nulo, ‘arr’ y ‘cadena’; se declaran las variables de tipo entero N con valor de 26, M con 100, k con 0 y q sin valor; se declara el carácter ‘letra’ de contenido ‘A’.

   Se reserva memoria para ‘arr’ de N+1 espacios para un arreglo de tipo carácter. Se inicia un bucle con k=0, mientras k sea menor a N, y al final de cada iteración k aumenta en 1; dentro del bucle, a la posición k del arreglo ‘arr’ se le asigna la suma de ‘letra’ con el valor de k (‘A’ + k); esto haría que a la casilla k se le asigne una letra del alfabeto en mayúsculas.

   Una vez terminado el ciclo, en la casilla de ‘arr’ 26 se le coloca el fin de cadena. Es iniciado otro ciclo con k=0, mientras k sea menor a N, y al final de cada de iteración k aumenta en 1; dentro del ciclo, se imprimirá el carácter de la posición k del arreglo ‘arr’, esto sería el abecedario. Al finalizar el ciclo, se imprime la cadena completa de ‘arr’, que sería nuevamente el abecedario en mayúsculas.

   En otro ciclo, con q=0 y k=0, mientras k sea menor a N, k aumentará en 2 y ‘q’ en 1 al final de cada iteración. Dentro de este bucle, en la posición ‘q’ del arreglo ‘cadena’ se le asigna la casilla k de ‘arr’; en este caso, la longitud del arreglo ‘cadena’ sería N/2 + 1, puesto que k aumenta de dos en dos, y el +1 es por el final de cadena. En la casilla ‘q’ de ‘cadena’ se le

coloca del final de cadena y se imprime la cadena completa de ‘cadena’, siendo esta “ACEGIKMOQSUWY”.

Comienza otro bucle con q=0 y k=N-1, mientras k sea mayor o igual a 0, k se restará en 6 y q aumentará en 1 al final de cada iteración. La operación que realiza este bucle es que en la casilla ‘q’ de ‘cadena’ se le coloque el carácter de la posición k de ‘arr’. Después del bucle, se le pone el fin de cadena a ‘cadena’ en la casilla ‘q’ y se imprime la cadena entera, la cual es “ZTNHB”.

El ultimo ciclo que se tiene es con q=0, mientras q sea menor a N, q aumentará en 1 por cada iteración; en la celda q del arreglo arr se coloca el carácter de la celda q+1 del mismo arreglo, esto hace que la celda q tenga el carácter de la siguiente casilla. Imprimiendo el arreglo da “BCDEFGHIJKLMNOPQRSTUVWXYZ”. Finalmente, se libera la memoria de los arreglos con free().

2. En la función ‘realizarAcrobacias’ se tiene que el parámetro ‘x’ es paso por valor, *q por referencia, ‘y’ por valor, *p por referencia & *w por referencia.

   En la función principal se tienen las variables a, b, c, Alpha, omega y delta, con valores de 100, 100, 6, 6, 7 y 3, respectivamente. A la variable b se le asigna el valor devuelto por la función ‘realizarAcrobacias’, con parámetros de ‘x’ siendo ‘a’, ‘q’ teniendo la dirección de Alpha, ‘y’ teniendo una copia del valor de ‘c’, p y w tienen la dirección de omega y delta, respectivamente.

   Dentro de la función ‘realizarAcrobacias’, encontramos un ciclo con un contador k=0, mientras k sea menor a x (= a= 100), y al final de cada iteración k se suma con *w (=delta = 3); en el ciclo, se tiene como primera instrucción asignar a una variable ‘c’ el valor de 0, luego, se inicia otro bucle que se repetirá hasta que una instrucción lo termine; dentro de este bucle se tiene la condición de que c sea menor a ‘y’ (=c = 6), si esto es cierto, entonces se termina el bucle; si no es verdadero, entonces se continúa con comprobar que el residuo de k/2 sea 0, si es así, entonces *q se sumará con k; si no, entonces *p se sumará con k, luego, fuera de la condición, c aumentará en 1. Puesto que la primera condición es verdadera, se termina el bucle y se procede a repetir el ciclo hasta que k supere ‘x’; después, se asigna a ‘x’ el valor de -600, para luego regresar el valor de la resta de ‘y’ & ‘x’, siendo esto 6-(-600) =606.

   Por lo tanto, devuelta en la función principal, tenemos que se imprime “a= 100, b=606, c= 6”, luego se imprime “Alpha= 6”, “beta= 3”, “omega= 7”. Los valores que pasaron por referencia se mantuvieron iguales porque no fueron modificados.

3. En la función principal, se crean tres apuntadores de arreglos de tipo entero, *arr10, *arrr100 y *arrr1000 con valor nulo; cada uno para la longitud de arreglo N= 10, 100, 1000, respectivamente.

Luego, para los tres arreglos (sin apuntador) se le asigna la invocación de la función crearArreglo con parámetro 10, 100 y 1000, respectivamente. Ahora se requerirá una función para inicializar las celdas del arreglo con ceros y otro para desplegarlos.

Empezamos con la función inicializarArreglo, con parámetros de un apuntador de arreglo de tipo entero y la longitud del arreglo N; en esta función habrá un ciclo con un contador k=0, mientras k sea menor a N, k aumentará en 1 al final de cada iteración; dentro del ciclo, en la celda k del arreglo se le asignará un 0. Esto sería todo para inicializar el arreglo.

Ahora, creamos una función desplegarArreglo que tendrá como parámetro un apuntador a arreglo entero, este será el arreglo por imprimir, y otro parámetro de un número entero que será la longitud N. Esta función tendrá un ciclo con un contador k=0, mientras k sea menor a N, k aumentará en 1; la única instrucción de este bucle será imprimir la celda k del arreglo.

Una vez teniendo las funciones, se invocan después de crear el arreglo, primero la función inicializarArreglo, luego la función desplegarArreglo; se invocarán estas dos funciones tres veces, cada una para cada valor de N. Finalmente, invocamos la función destruirArreglo para liberar la memoria del arreglo que esté como parámetro; esta función se invocará igualmente tres veces, cada vez para un arreglo distinto.

4. La función para crear un arreglo dinámico será *crearArreglo, de tipo double y tendrá como parámetro la longitud N deseada; dentro de la función, se declara un apuntador a arreglo *arr de tipo de dato double con valor nulo. Después, se comprueba que la longitud N no sea menor o igual a 0; a continuación, al arreglo arr se le reserva memoria con malloc, la longitud N multiplicada por el tamaño del tipo de dato double; luego, se regresará el arreglo como valor de retorno.

   Para inicializar el arreglo, creamos la función inicializarArreglo que no regresará nada, pero tendrá como parámetro un apuntador a un arreglo de tipo double y la longitud N del arreglo. En la función, se comprobará que el arreglo no tenga valor nulo, si llegase a tenerlo, se terminaría la función; si no, se procede con un ciclo con un contador k=0, mientras k sea menor a N, k aumentará en 1 al final de cada iteración; en este bucle, en la celda k del arreglo se le asignará un 0. Una vez termine el ciclo, ya se habría realizado la tarea de la función.

   Para desplegar el contenido, será una función que no devolverá nada y se llamará desplegarArreglo, cuyos parámetros serán: un apuntador a un arreglo de tipo double y la longitud N; con un ciclo de contador k=0, mientras k sea menor a N, k aumentará en 1 en cada iteración, se imprimirá el contenido de la celda k del arrelgo.

   Para destruir el arreglo, creamos la función destruirArreglo y tendrá como parámetro el apuntador a arrelgo de tipo double y la única instrucción que tendrá es la función free() que liberará la memoria del parámetro.

Suponiendo que el arreglo ya posee números en todas sus celdas, para calcular la suma de estos números basta con una función que llamaré calcularSuma, la cual devolverá un valor de tipo double y como parámetro será un apuntador a un arreglo de tipo double y la longitud N del arreglo. Declaramos una variable para la suma con valor de 0, y se verificará que el arreglo no tenga valor nulo, si fuera a tenerlo, se termina la función; si no lo tiene, a continuación, en un ciclo con un contador k=0, mientras k sea menor a N, k incrementará en uno tras cada iteración; aquí, la variable suma se sumará con sí misma y con el contenido de la casilla k del arreglo. Después de terminar el ciclo, se tendrá como valor de retorno la variable suma.

Para el promedio, se usará una función que llamaré calcularPromedio y como parámetros será la suma de los números y la longitud N. Se declara una variable promedio y ésta será igual a la suma dividida entre N. Se retorna el valor de promedio.

5. Suponiendo que el arreglo ya fue creado previamente, para ordenar un arreglo, se invocará la función OrdenarArreglo, la cual tendrá como parámetro un apuntador a arreglo de tipo entero y la longitud del arreglo. Dentro de la función, se verifica que el arreglo no tenga valor nulo, si llegase a tenerlo, se acaba la función; si no, procede a realizar una consulta de si el ordenamiento será ascendente o descendente; cualquiera de los dos tipos de ordenamientos, se declara una variable aux y se inicia un bucle con un contador i=0, mientras i sea menor a N, i aumenta en uno tras cada iteración; en este bucle, habrá otro ciclo con un contador j=i+1, mientras j sea menor a N, j incrementará en 1 al acabar cada iteración; aquí, habrá una condición que dependerá si se trata de un orden ascendente o descendente. Para el orden ascendente, se comprobará que el número arreglo en la celda j del arreglo sea menor al número de la casilla i del mismo arreglo. Si esto es verdadero, aux toma el valor del número de la celda j del arreglo, luego, la casilla j obtendrá el valor de la casilla i del arreglo, después en la posición i se tendrá como nuevo valor el contenido de aux. Si no se cumple, entonces se sigue haciendo el segundo bucle.

   Para el orden descendente, la condición cambia con que se comprobará que la celda j del arreglo sea mayor a la posición i del mismo arreglo. Después, el intercambio de valores será el mismo.

   Para desordenarArreglo, que tiene como parámetros un apuntador a arreglo de tipo entero (este será el arreglo inicial) y la longitud N, se declararán dos apuntadores a arreglos de contenido nulo, uno que almacenará las celdas repetidas y otro que tendrá los números desordenados. Se realiza una verificación de que el arreglo del parámetro no tenga valor nulo, si fuera a tenerlo, se termina la función; si no, continúa con la reserva de memoria para los arreglos será con malloc, con longitud de N+1 y N, respectivamente. Luego, en el arreglo de números desordenados, en la casilla 0, se le colocará el contenido de la celda 0 del arreglo de enteros; después, se inicia un ciclo con un contador i=0,

mientras i sea menor a N, i incrementa en 1 por iteración; dentro del ciclo, con una variable k, se le asignará un valor aleatorio entre 0 y N.

Después, se iniciará otro bucle con un contador j=0, mientras j sea menor o igual a i, j aumentará en uno cada que vez finalice una iteración; dentro de este segundo bucle, habrá una condición que comprobará que la celda 0 del arreglo de celdas repetidas sea igual a k=0, si es así, entonces en el arreglo de números desordenados se almacenará el número de la casilla k del arreglo inicial. Si no, pasará a otra condición en la que se comprobará que la posición j del arreglo de celdas repetidas no sea igual al número aleatorio de k. En caso de que no sean iguales, en la casilla i del arreglo de números desordenados será almacenado el número de la posición k del arreglo inicial, y se guardará el número de k en la casilla i+1 del arreglo de celdas repetidas, de esa forma se evitará que se repitan los números cuando no se cumpla la condición. En caso de que sí sean iguales, entonces la casilla i del arreglo de números desordenados será 0, en la posición i+1 del arreglo de celdas repetidas será 0, ya que no se aceptan números repetidos, e i se restará en una unidad para que se repita el ciclo con las mismas casillas, pero con un diferente valor de k. Al final de los bucles, se imprimirá el arreglo desordenado.

6. Se declaran 4 arreglos de tipo carácter longitud 60, linea0, linea1, línea2, linea3; también se declara un apuntador a arreglo de tipo carácter *rayuela con valor nulo. Se crean las celas del arreglo con la función crearArreglo, el cual tendrá como parámetro la longitud del arreglo, que es 339, y usará una condición para asegurarse que la longitud no sea igual o menor a 0 y la función malloc para reservar la memoria necesaria.

   Se invoca la función concatenar, cuyos parámetros con apuntadores a arreglos; el primero es *nuevo, el cual será el arreglo final; el segundo es *prefijo, el cual es la cadena con que se iniciará el nuevo arreglo; el tercero es *sufijo, el cual será la cadena con que se finalice la cadena del nuevo arreglo. En la primera invocación, se pasan como parámetros rayuela, linea0, linea1; dentro de esta función, se declaran las variables k y q, luego se comprueba que ninguno de los tres valores tenga como valor nulo, si llegase a ser así, entonces se termina la función. Si todo está en orden, entonces se procede a iniciar un ciclo con k=0 y q=0, mientras no se encuentre el final de cadena del arreglo prefijo (linea0), k y q incrementarán en 1 tras cada iteración. Dentro de este bucle, en la celda q del arreglo nuevo (rayuela) se le asigna el carácter de la casilla k del arreglo prefijo. Después de que finalice el ciclo, se inicia otro, pero solo con el contador k=0, mientras no se encuentre el fin de cadena del arreglo sufijo (linea1), k y q aumentarán en 1 al acabar una iteración (el valor de 1 se conversa); dentro de este bucle, en la celda q del arreglo nuevo se le asigna el carácter de la casilla k del arreglo sufijo. Cuando termine el bucle, en la celda q del arreglo nuevo se le coloca el fin de cadena.

   Luego, se invoca la función relatar, cuyo parámetro es un apuntador a arreglo, en este caso será el arreglo rayuela. Esta función se encarga de imprimir el contenido del arreglo, pero antes de eso verifica que el parámetro no tenga valor nulo.

Se repite el proceso de invocar la función de concatenar con las demás líneas (linea1 con linea2; linea2 con linea3) y cada invocación de concatenar tendrá también una invocación de la función relatar; de esta forma se van a concatenar las cadenas y se imprimirán.

Después de eso, se invoca la función redactar, cuyos dos parámetros son apuntadores a arreglos de tipo carácter, el primer es *texto y el segundo es *cadena. En esta función, se declara la variable k y q; luego, se comprueba que texto y cadena tengan valor nulo, si esto es cierto, se termina la función; si no es cierto, entonces se procede a iniciar un ciclo con el contador q=0, mientras no se encuentre el fin de cadena del arreglo texto (rayuela), q incrementa en 1 por cada iteración; eso sería todo lo que haría el primer bucle. Después, se inicia otro bucle con el contador k=0, mientras no se encuentre el fin de cadena del arreglo cadena (linea0), k y q aumentan en 1 por cada iteración. Dentro de este bucle, la celda q del arreglo texto se le asigna el contenido de la casilla k del arreglo cadena; al finalizar el bucle, en la posición q del arreglo texto se le coloca el fin de cadena. Lo que hace esta función es añadir el contenido del arreglo linea0, linea1, linea2 y linea3 al final del arreglo rayuela.

Esta función de redactar se invoca para cada arreglo de linea como segundo parámetro; después de esto, es invocada la función relatar para imprimir el contenido del arreglo. Finalmente, se libera la memoria del arreglo rayuela con la función destruirArreglo.

7. Para crear una cadena de N caracteres, creamos una función que devolverá un arreglo de caracteres llamada *crearCadena, cuyo único parámetro será los N caracteres; dentro de esta función, se declara un apuntador a arreglo de tipo carácter llamado *arr con valor nulo, después, se comprueba que N no sea menor o igual a 0, y con la función malloc, se reserva memoria con el parámetro N+1 (una celda extra para el fin de cadena) y el tipo de dato del arreglo. Luego, se utiliza el arreglo como valor de retorno.

   Para destruir una cadena, simplemente creamos una función que llamada destruirCadena con un único parámetro que es un apuntador a arreglo de tipo cadena; en la función, se utiliza la función free() para liberar la memoria del arreglo (que estará como argumento de la función).

   Para desplegar una cadena, declaramos una función desplegarCadena con único parámetro un apuntador a arreglo de tipo cadena; se comprueba que el arreglo tenga valor nulo, si es así, se acaba la función; si no, se procede a empezar un bucle con un contador k=0, mientras en la celda k del arreglo no se encuentre el fin de cadena, k incrementa 1 por iteración. En la función, se imprime la celda k del arreglo.

8. Copiar una cadena fuente a una cadena destino: la función copiarCadenas devolverá un valor entero para indicar que la operación tuvo éxito. Esta función tendrá como parámetros los apuntadores a arreglo *destino (cadena resultante de la copia) y *fuente (cadena que será copiada); primero, se verifica que ambos arreglos no tengan valor nulo, si fueran a tenerlo, la función retorna ERROR (= 0) y termina; si no, entonces se comienza

un ciclo con un contador k=0, mientras no se encuentre el fin de cadena, k aumenta 1 por iteración. Se realiza la operación de asignar a la celda k del arreglo destino el carácter de la casilla k del arreglo fuente. Cuando termine el ciclo, la función dará como valor de retorno ÉXITO (= 1).

Concatenar dos cadenas: la función concatenarCadenas tendrá tres parámetros de apuntadores a arreglo de caracteres, *sol (cadena concatenada), *cada (primera cadena para concatenar), *cadB (segunda cadena para concatenar). Se verifica si estos tres arreglos tienen valor nulo, si es así, se acaba la función con valor de retorno ERROR (= 0); si no es el caso, se procede a empezar un bucle con dos contadores k=0 y q=0, mientras en la celda k de cada no se encuentre el fin de cadena, k y q aumentan en 1 por ciclo; se realiza una asignación de valor, la celda q de sol recibe el contenido de la celda k de cada. Después de terminar el ciclo, se inicia otro, pero sólo con el contador k=0, teniendo la misma condición, pero esta vez para cadB, k aumenta en 1 por ciclo; ahora, en la casilla q (todavía mantiene su valor del primer ciclo) del arreglo sol se le asigna el contenido de la posición k del arreglo cadB. Cuando finalice el ciclo, la función dará como valor de retorno ÉXITO (= 1).

9. Antes de empezar, se considera que la cadena existe y tiene caracteres como contenido.

   Encontrar la longitud de la cadena: con una función obtenerLongitudCadena, con único parámetro un apuntador a arreglo de tipo carácter *cadena; primeramente, se comprueba si el arreglo cadena tiene valor nulo, si es el caso, acaba la función; si no lo tiene, procede a comenzar un bucle con un contador k=0, mientras no se encuentre el fin de cadena del arreglo cadena, k aumenta en 1 por cada ciclo. El bucle no hará nada, solo aumentará el contador k y al finalizar el bucle retorna el valor de k.

   Buscar un carácter de la cadena: creando la función buscarCaracter, el cual tendrá dos parámetros, uno es un apuntador a arreglo de caracteres *cadena y otro es un carácter común y corriente. Se verifica si cadena tiene valor nulo, si es así, la función acaba; si no, entonces se procede a comenzar un bucle con un contador i=0, mientras no se encuentre el fin de cadena del arreglo cadena, i se sumará en 1 por cada iteración; dentro de este bucle, se tendrá otro con un contador j=0, mientras no se encuentre el fin de cadena del arreglo carácter, j incremente en 1 por ciclo. Habrá una condición que comprobará si la celda i del arreglo carácter no es igual a la celda j del arreglo cadena, si se cumple este, se termina el segundo bucle; si no, se procede a otra condición aparte. La otra condición consiste en que, si la celda j del arreglo carácter es el final de cadena, entonces se da como valor de retorno el valor de i. En el caso de que no se encuentre una coincidencia exacta, el valor de retorno será -1.

10. Comparar dos cadenas: creando la función compararCadenas con dos parámetros de apuntadores a arreglos de tipo carácter llamados *cadena (primera cadena) y *cadena (segunda cadena). Se declara una variable auxiliar que ayudará para conocer cuántas diferencias hay entre ambas cadenas, esta variable tendrá un valor inicial 0. Se comprueba

que ambos arreglos no sean nulos, si fuera a tener ese valor, la función terminaría. Para que dos cadenas sean exactamente las mismas, las posiciones de los caracteres de ambas cadenas deben ser las mismas; por lo tanto, para lograr esto, se usará un ciclo con dos contadores i=0 & j=0, mientras no se encuentre el fin de cadena del arreglo cadena o el del arreglo cadenaB, ambos contadores aumentarán en 1 por cada ciclo; habrá una condición que comprobará si la celda i del arreglo cadena no es igual a la celda j del arreglo cadenaB, si esto es verdadero, se procede a incrementar de valor en 1 a la variable auxiliar; si no fuera el caso, continuaría el bucle hasta su fin. Antes de que termine la función, ésta devolverá el valor de la variable auxiliar.

Invertir la cadena: creando una función llamada invertirCadena se logrará el cometido; recibirá dos parámetros de apuntadores a arreglos de tipo carácter, *inversión y *original. Se declara una variable L para almacenar la longitud de la cadena original; luego, se comprueba que el arreglo original tiene valor nulo, si fueran a tenerlo, la función da como valor de retorno 0 y termina la función. Ahora, con un ciclo de contador k=0, mientras no se encuentre el fin de cadena del arreglo original, k aumenta 1 por iteración; el valor de k es guardado en L. Después, en otro bucle, se usa el mismo contador k=0, mientras k sea menor a L, a k se le suma 1 tras iteración. En la celda k del arreglo inversión se le colocará el contenido de la celda L- (k+1) del arreglo original, esto se debe a que se le asigna el contenido de la última casilla y se quiere evitar el fin de cadena. Después del fin del bucle, se coloca el fin de cadena en la celda L del arreglo inversión y la función devuelve el valor de 1 para demostrar que la operación se realizó con éxito.

11. Representar número entero en una cadena de caracteres: considerando que no se le ha reservado memoria para el arreglo debido a la variante longitud del número, en una función aparte, el cual tendrá el apuntador a arreglo y el número, se realizará un ciclo con un contador i=0, mientras el número sea mayor a 0, i aumenta en uno por cada ciclo; dentro, se hará la operación de que el nuevo valor del número será la de su división entre 10, de esta forma, se sabrá la longitud del número gracias al contador i; después, con malloc se reservará la memoria para el arreglo, cuyo longitud será de i+2 (por el signo y el fin de cadena). Finalmente, la función regresa el arreglo.

    Con lo anterior, a continuación, se crea la función convertirEnteroACadena, con dos parámetros: un apuntador a arreglo de tipo carácter *número y un número entero n. Se declara una variable auxiliar y se verifica si el arreglo tiene valor nulo, si es así, regresa ERROR (= 0); si no es así, se procede a comprobar que el número n sea mayor a 0, si es así, en la celda 0 se lo coloca el signo ‘+’, si no, se coloca el signo ‘-‘. Luego, se inicia un bucle con un contador i=1, mientras n sea mayor a 0, i aumenta en 1 por iteración; en la celda i del arreglo numero se le colocará el residuo de la división de n/10, luego n se le asigna el cociente de la misma división; al finalizar el bucle, se coloca el fin de cadena en la celda i. Después, se declara la variable l para recibir la longitud de la cadena, puesto que el bucle anterior hizo ese trabajo de forma implícita y así no será necesario hacer otro bucle. Tras eso, comenzamos otro bucle que tendrá dos contadores, i=1 & j= l - 1, mientras i sea menor a l-1 (la celda l tiene el fin de cadena) y mientras j sea mayor a 0 (en la celda 0 está

el signo), i incrementa en 1 y j se resta en 1; la variable auxiliar obtendrá el contenido de la celda j del arreglo, luego, la celda j cambia su contenido por el de la celda i y, finalmente, la celda i del arreglo recibirá el contenido de la variable auxiliar. Con el bucle terminado, se imprime el contenido del arreglo y se da como valor de retorno ÉXITO (= 1)

Número representado por una cadena de caracteres: creando la función convertirCadenaANumero, que tiene dos parámetros: un apuntador a un número entero *n (resultado) y un apuntador a arreglo de tipo carácter *numero (entrada); se declara una variable auxiliar con valor de 0, y se comprueba que el arreglo numero no tenga valor nulo, si fuera a tenerlo, la función da como valor de retorno ERROR (= 0) y termina. Si no, entonces se procede a conocer la longitud del arreglo con un ciclo cuyo contador seguirá sumando mientras no se encuentre el fin de cadena del arreglo, luego, declarando una variable para l para la longitud, ésta recibe el valor del contador del último ciclo menos 2 (para no incluir la celda del signo y fin de cadena). A continuación, se comprueba si la celda 0 del arreglo es igual a '-', si es así, al número n se le da el valor de -1; si no, eso significa que el número es positivo, puesto que, generalmente, a los números positivos no se les agrega el signo, así que a n se le da valor de 1. Ahora, se comienza un bucle con i=1, mientras no se encuentre el fin de cadena del arreglo, i aumenta 1 por cada iteración (el contador inicia en 1 porque en la celda 0 está el signo del número); el valor de la variable auxiliar es la suma de éste mismo con el resultado del producto de la resta de la celda i del arreglo con 48 por 10 elevado a la l, y l se resta en 1. Después, del bucle, n obtiene el resultado del producto de n con la variable auxiliar, y finalmente la función devuelve ÉXITO (= 1) para indicar que la operación se realizó correctamente. En la función principal se imprimirá el resultado de n.

12. Se declaran 3 matrices con doble apuntador y valor nulo; se tiene que el alto M y el largo N es 3. A cada matriz se le invoca la función **crearMatriz, la cual crea la matriz con malloc, reservando memoria para M celdas, después, con un bucle de i=0, mientras i sea menor a M, i aumenta en 1, se reserva memoria para N celdas en la casilla i del arreglo; al finalizar ese bucle, comienza otro en el que se le colocan números en cada celda de la matriz. Todas las matrices terminan con el mismo contenido, el cual es: primera fila de ceros; segunda fila con 0, 1, 2; tercera fila 0, 2, 4.

    Posteriormente, se invoca la función sumarMatrices, los cuales pasan como parámetros la matrizC, matrizA, matrizB, M y N; se comprueba si alguna de las matrices tiene valor nulo, si es así, se termina la función; si no, se procede a iniciar un bucle con i=0, mientras i sea menor a M, i incrementa en 1. Se ejecuta otro bucle dentro de este último con j=0, mientras j sea menor a N, j aumenta en 1; en este segundo bucle, en la celda (i, j) de la matrizC se coloca la suma del contenido de la celda (i, j) de la matrizA con el de la celda (i, j) de la matrizB. Se finaliza con que el contenido de la matrizC es: primer fila llena de ceros; segunda fila con 0, 2, 4; tercera fila con 0, 4, 8.

    Antes de finalizar, es invocada la función multiplicarMatrices, con los mismos parámetros que la función anterior; igualmente se comprueba si alguna de las matrices tiene valor

nulo, si es así, acaba la función. Si no, entonces se inicia un ciclo triple, el primer con i=0, mientras i sea menor a M, i aumenta en 1; el segundo bucle tiene j=0, mientras j sea menor a N, j incrementa en 1. Como primera instrucción se le coloca en la casilla (i, j) de la matrizC el número 0, de esta forma la matrizC se llenará de ceros; luego, se comienza el tercer bucle con k=0, mientras k sea menor a N (puesto que no se ha declarado ninguna otra variable para la longitud de la matriz), k se suma en 1. Dentro de este bucle, la celda (i, j) de la matrizC recibe la suma del de misma celda con el producto del contenido de la casilla (i, k) de la matrizA por el de la casilla (k, j) de la matrizB. Después del ciclo triple, el contenido de la matrizC es: primera fila llena de ceros; segunda fila con 0, (0 + 1*1 + 2*2=) 5, (0 + 1*2 + 2*4=) 10; tercera fila con 0, (0 + 2*1 + 4*2 =) 10, (0 + 2*2 + 4*4=) 20. Finalmente, se libera la memoria de esas matrices invocando destruirMatriz y usando la función free().

13. Se comienza con que 'p' tiene la dirección de 'x' #D004, así que *p es 2020; después, 'hyper' tiene la dirección de 'p' #B002, por lo que *hyper hace referencia a la dirección de 'x' #D004 y *(*hyper) tiene valor de 2020. Luego, 'q' tiene la dirección de lo que hace referencia *hyper (dirección de 'x' #D004) y, a su vez, conoce a qué hace referencia 'p' (dirección de 'x' #D004); *q se le cambia de valor a 2021, por lo que *(*hyper), *p y 'x' también cambian a ese valor.

14. Antes de la función principal, se tiene una estructura "complejo" con lo necesario para formar un número complejo; después de la estructura, se define *Complejo (para apuntadores) y t_complejo (para variables normales).

    En la función principal, se declaran los apuntadores a estructura a, b y c con valor nulo; en a y b se les declara la función crearNumeroComplejoCartesiano con parámetros re (parte real) e im (parte imaginaria) y crearNumeroComplejoPolar con rho (parte real) y phi (parte imaginaria), respectivamente.

    En la función para el número complejo cartesiano, se declara la estructura con apuntador z con valor nulo, y las variables rho y phi; se reserva memoria para la estructura con malloc y de tamaño de la estructura estática (t_complejo). Se invoca la función convertirPolar, con parámetro la dirección de rho, la dirección de phi, una copia de re e im; la función convertirPolar se dará valor a rho y phi. Para obtener rho, se obtiene la raíz cuadrada de la suma de los cuadrados de re e im; para phi, se calcula el arco tangente de la división de im entre re. Posteriormente, se tiene que en la estructura z en su variable real se le asigna el valor de re, para la variable imaginario tiene el contenido de im, para la magnitud es rho y para la fase es phi; para terminar la función, se retorna z, puesto que la función devuelve una estructura.

    Con la función del número polar, es similar a la anterior, solo que esta vez recibe como parámetro el valor de rho y phi, además de que en la función se declaran las variables re e im. Después de la reserva de memoria, se invoca la función convertirCartesiano, el cual recibe como parámetros la dirección de re e im, una copia de rho y phi; esta función hace

que re sea el producto de rho por el coseno de phi, e im es el producto de rho por el seno de phi. Regresando a la anterior función, a la variable real de la estructura z se le da el valor de re, a imaginario el de im, el de magnitud, rho, y la fase, phi. Devuelve la estructura z.

De regreso a la función principal, se invoca la función SumarComplejos, con parámetro la estructura a y b. En esa función, se declara un apuntador a estructura w con valor nulo y se reserva memoria para la estructura w; luego, se verifica que las estructuras no tengan valor nulo, si lo tienen, la función se termina; si no, se prosigue a dar valores a las variables de la estructura w. Para su variable real, se realiza la suma de la variable real de a y b; para su variable imaginario, se hace la suma de la variable imaginario de a y b. Posteriormente, se invoca la función convertirPolar, que recibe como parámetro la dirección de la variable magnitud y fase de la estructura w, y una copia de la variable real e imaginario de la estructura w. Después, de la conversión, se devuelve la estructura w como valor de retorno, es decir, c tiene el contenido de esta estructura.

Después, se despliega el contenido de la estructura c con la función desplegarNumeroComplejo. Luego, a c se le invoca la función multiplicarComplejos, que recibe como parámetro la estructura a y b, en donde se realizan instrucciones similares al de la función sumarComplejos, la variación es que la variable magnitud de la estructura w es la multiplicación de la variable magnitud de la estructura de a por la de b; la variable fase de la estructura w es la suma de la variable fase de la estructura a y b; después, se invoca la función convertirCartesiano, cuyo parámetro es la dirección de la variable real e imaginaria de la estructura w, y una copia de la variable magnitud y fase de la estructura w. Finalmente, la función devuelve la estructura w como valor de retorno.

Se imprime el contenido de la estructura c con desplegarNumeroComplejo, y finalmente se libera la memoria de las estructuras que se crearon con destruirComplejo usando la función free.

15. Se define la estructura ‘complejo’, que tiene las variables: real, imaginario, rho y phi; también se tiene t_complejo y el apuntador *Complejo para usar la estructura rápidamente.

    En la función principal declaramos un apuntador a estructura con ‘Complejo’ que llamaré ‘a’ y tendrá valor nulo, esto servirá como arreglo; también tendremos el número N que serán las raíces del número complejo. En la variable ‘a’ se le invoca la función crearArregloComplejo, cuyo parámetro es N; con esta función, ‘a’ se volverá un arreglo de estructuras, en donde cada celda es una estructura idéntica a las otras, ya que esas estructuras se formaron con la función crearNumeroComplejoCartesiano, con sus parámetros re e im siendo cero. Esta función también invoca una función para obtener la representación polar de los números cartesianos.

Posteriormente, se invoca la función calcularRaicesComplejas, que tiene como parámetro un apuntador a estructura *raíces (gracias a esto no es necesario devolver la estructura como valor de retorno) y una variable entera N; la primer será el arreglo y el segundo son las raíces del número complejo. Se declara una variable phi y se comprueba si el arreglo raíces tiene valor nulo, de ser así, se termina la función; continuando, phi adquiere el resultado de multiplicar 2.0 por PI dividido entre N (convertido a tipo de dato flotante). Luego, se inicia un bucle con k=0, mientras k sea menor a N, k se suma en 1; en este bucle, en la celda k del arreglo raíces recibe la estructura creada por la función crearNumeroComplejoPolar, con sus parámetros siendo que rho es 1.0 y phi es 0.0, esta función da un nuevo valor a los atributos de la estructura de la celda k del arreglo raíces, al igual que convierte los números polares a su representación cartesiana. Después, en la celda k de raíces, en su variable fase, se le asigna el valor de multiplicar k por phi. Finalmente, se invoca la función convertirCartesiano, el cual recibe como parámetro la dirección de la celda k de raíces en su atributo real e imaginario, y también una copia del contenido de la variable magnitud y fase de la celda k del arreglo raíces.

A continuación, en la función principal, con una función desplegarArregloComplejo, se imprime el contenido del arreglo usando un ciclo con k=0, mientras k sea menor a N, k incrementa en 1; se imprime el contenido de la variable real, imaginario, magnitud y fase de la celda k del arreglo (sin usar apuntador). Finalmente, con la función destruirArregloComplejo, que tiene de parámetro un apuntador a estructura *arr y una variable entera N; se verifica si arr es nulo, si es así, se termina la función. Si no, en un ciclo con k=0, mientras k sea menor a N, k aumenta en 1, se verifica si la celda k de arr no es nulo, si no lo es, se libera la memoria de la celda k de arr. Después del ciclo, se libera la memoria de arr.

16. Se define una estructura ‘codice’, con los atributos *dígitos (un arreglo), base, decimal y N, todos de tipo entero; también se definen dos tipos de estructuras, uno con apuntador *Codice y otro normal t_codice. En la función principal se declaran dos estructuras, ‘codex’ y ‘escrito’, ambas con valor nulo. Luego, se declara un arreglo estático ‘sec’ de 8 celdas con los números 1, 3, 2, 0, 1, 1, 1, 3; también se tiene una variable num=2019 (número en decimal), b=4 (la base numeérica) y L=8 (longitud).

    Se invoca la función crearCodiceBase, el cual se le pasa como parámetro la dirección de codex (= *code), el arreglo sec (= arr), una copia de b (= base) y L (= N). En esa función, se declara un apuntador a estructura ‘glifo’ con valor nulo; a esa estructura se le reserva memoria con malloc, con tamaño de t_codice. Después, el arreglo dígitos de glifo se le reserva memoria para N celdas y en esas celdas se les coloca ceros con la función crearArreglo; luego, con un ciclo con k=0, mientras k sea menor a N, k aumenta en 1, y en la casilla k del arreglo dígitos de la estructura glifo se le coloca el contenido de la celda k del arreglo arr (sec). Posteriormente, al atributo base y N de glifo se lo da el valor de base (b) y N (L), respectivamente. Antes de finalizar, a la variable decimal de la estructura glifo se le invoca la función convertirBase10, pasando como parámetro el arreglo arr, base y N. En eta función, se tiene declarada una variable x=0 que servirá para almacenar el número

decimal; se verifica si arr tiene valor nulo, de ser así, se termina la función; si no, se procede a un ciclo con k=0, mientras k sea menor a N, k incrementa en 1 y, en el ciclo, x obtiene el valor de la suma de sí mismo con el producto de la celda k del arreglo arr por la base elevado a la k (el digito menos significativo es el de la celda 0). Tras finalizar el ciclo, la función regresa el valor de x, siendo que la variable decimal de glifo tenga ese valor. Para terminar, *code se vuelve la estructura glifo, es decir, codex tiene todos los datos de glifo.

En la función principal, se invoca la función desplegarCodice para que imprima el contenido de la estructura codex. Después, se invoca la función crearCodiceDecimal, pasando como parámetro la dirección de la estructura escrito (*code), y una copia de num (dec), b (base) y L (N); se declara un apuntador a estructura 'glifo', se reserva memoria de tamaño de t_codice, y se reserva memoria y se asignan valores al arreglo dígitos de glifo con la función crearArreglo. Luego, a la variable decimal, base y N de glifo adquieren el valor de dec (num), base (b) y N (L), respectivamente. Tras eso, es invocada la función convertirBaseB, teniendo como parámetro el arreglo dígitos de glifo (no es la dirección de memoria ya que se trata de un arreglo), dec (x), base y N; esta función convertirá el número decimal a la base numérica. Se tienen las variables lim, k, q y r, las tres siendo cero, y la variable dec siendo x. Se comprueba si el arreglo tiene valor nulo, si es así, se termina la función; si no, se procede a un bucle en que dec será la división de sí mismo con la base, luego, lim aumentará en 1 y se comprobará si dec es 0, si es así, se romperá el bucle. Luego, se comprueba si lim es mayor a N, si es así, acaba la función; si no, dec vuelve a tener el valor de x y se inicia otro bucle. En este bucle, q recibe el resultado de dividir dec entre base, luego, r tiene valor del residuo de esa división, la celda k del arreglo tendrá el valor de r (la celda 0 es el digito menos significativo) y k se sumará en 1; luego, se comprobará si q es igual a 0, si es así, se termina el ciclo, si no, dec recibe el valor de q. Puesto que se trata de un arreglo, no hace falta retornar un valor. Para finalizar, *code se le asigna el contenido de la estructura de glifo, así que la estructura escrito tiene ese contenido y luego se imprime con la función desplegarCodice.

Para imprimir el arreglo de la estructura con solo pasar la estructura como parámetro, se usa el atributo N de dicha estructura como el limite del contador del ciclo. Además, imprimir el arreglo es por medio de la celda k del arreglo de la estructura (la estructura no es un arreglo).

Finalmente, se invoca la función destruirCodice, que recibirá la dirección de la estructura que se quiera liberar memoria. En esa función, se crea un apuntador de estructura q con valor nulo, esta estructura obtiene la dirección del parámetro (y al tener la dirección, los cambios de q afectarán al parámetro); se verifica si q es nulo, si es así, se termina la función. Si no, se comprueba si el arreglo de la estructura es nulo, si es así, se invoca la función destruirArreglo, cuyo parámetro es el arreglo de la estructura; en esa función, se usa la función free para liberar la memoria del arreglo. Después, en destruirCodice, a las variables decimal, base y N de la estructura se les da valor 0 y se usa la función free para terminar de liberar la memoria de la estructura.

17. En la función principal, se declara un arreglo de caracteres que tendrá el nombre del archivo; también, una variable N, que será la longitud del arreglo, y un arreglo de enteros para almacenar la secuencia de números. Se le invoca la función 'leer' (*leer) al arreglo de enteros, pasando como parámetro el arreglo que contiene el nombre del archivo y la dirección de N; esta función devolverá un arreglo de enteros.

    En la función 'leer', se crea un apuntador de tipo FILE * de nombre 'archivo' (un fichero), también se declara un arreglo de enteros, y variables k, x y lim, siendo los tres 0. A 'archivo' se le invoca la función fopen, que tiene como parámetro el arreglo con el nombre del archivo y la letra "r" significa que se abrirá el archivo para su lectura. Después, se introduce un valor para lim, para luego asignar ese valor a *N; posteriormente, al arreglo de enteros se le reserva memoria de acuerdo con el valor de lim. De eso, se procede a un ciclo con k=0, mientras k sea menor a lim, k se suma en 1, y pedirá un valor para x con la función fscanf (primero recibe el nombre del fichero (archivo), se especifica el tipo de dato introducido y lo asigna a x); ese valor de x se colocará en la celda k del arreglo numérico, con esto se tendrá una sucesión de números. Después del ciclo, se cierra el fichero con fclose y como parámetro el nombre del fichero (archivo). Para finalizar, la función regresa el arreglo de enteros.

    De regreso a la función principal, se invoca la función 'escribir', que tendrá de parámetro el nombre del archivo, el arreglo de enteros y la variable N, la cual ya tiene un valor debido a la función anterior. En la función 'escribir', se crea un apuntador de tipo FILE *, con el nombre 'archivo', el cual es un fichero; además, se declaran dos variables enteras, k y x, con valor a 0. Al fichero 'archivo' se le invoca la función fopen para abrir el archivo cuyo nombre es el contenido del arreglo de caracteres y se realizará la operación "w" de escritura. Se imprime el valor de N, que es la longitud del arreglo de enteros; después, se inicia un bucle con k=0, mientras k sea menor a N, k incrementa en 1. En este bucle, la variable x adquiere el valor de la celda k del arreglo de enteros y se invoca la funcion fprintf para imprimir en el fichero 'archivo' y mostrar el valor de x; esto se repetirá hasta que k alcance a N. Finalmente, se cierra el fichero con fclose.

18. Se tiene que en la función principal se declara las variables de tipo carácter fuente, auxiliar y destino, con 'A', 'B' y 'C', respectivamente. También tenemos la variable N que será la cantidad de discos, y contador siendo 0. Se invoca la función moverDisco, teniendo como parámetro destino (dest), auxiliar (aux), fuente (fte), N (N), y la dirección de contador (*cont).

    En la función moverDisco, se verifica si N tiene valor de 0, si es así, se termina la función. Si no, se vuelve a invocar la función moverDisco, esta vez con que el parámetro dest será aux, aux es dest y fte es fte, N es N-1 y *cont no cambia. Esta función se seguirá repitiendo hasta que N sea 0; cuando suceda esto, se continúa la función moverDisco cuando N es 1 (cuando es 0 se termina la función), se imprime el valor del contador para conocer qué paso es, el número del disco y qué torre es la fuente y el destino. Debido a que cambian

los parámetros de la función, la fuente y el destino cambian constantemente. Luego, el contador *cont se suma en 1 y se vuelve a invocar la función moverDisco, cuyo parámetro es dest como dest, fte como aux y aux como fte, con N como N-1 y cont se mantiene. En la segunda vez que se invoca la función cuando N es 1, N es 0, por lo que la función con N=0 y N=1 termina, y pasa a N=2, en la cual se volverá a invocar la misma función, pero con distintos parámetros.

La recursividad de la funciona debido al cambio de lugar de los parámetros dest, aux y fte, y que las torres destino, auxiliar y fuente no siempre son las mismas, además de que existen patrones comunes en el movimiento de los discos.

19. En la función principal, se tienen 4 variables, N (nivel) y n (nodo) siendo 0, y lim= 4 (nivel límite) y base=3 (árbol ternario); se invoca la función arbolBinario, con sus parámetros siendo N (nivel), n (nodo) y lim (límite).

    En esta función, se declara una variable k para funcionar de contador. Se verifica si el valor de 'nivel' es mayor o igual a 'limite', si es así, se termina la función; si no, se inicia un bucle con k=0, mientras k sea menor al valor de 'nivel', k aumenta en 1, entonces se imprime un espacio (se imprimirá la cantidad de espacios dependiendo del valor de 'nivel', eso mostrará en qué nivel se está). Después, se imprime en qué nivel se encuentra y el nodo del árbol (su valor, en ambos casos); de ahí, pasa a invoca la función arbolBinario, esta vez con un valor adicional para 'nivel', el nodo tendrá un valor del doble y el limite se mantiene. Esta función hará que se impriman todos los nodos 0, para luego, cuando finalice la función, invocar de nuevo la misma función pero el nodo con valor del doble más 1 (2*nodo+1), esto hará que se impriman los niveles correspondientes de los nodos siguientes, hasta llegar al nivel 4; cuando llegue a nivel 4, se terminará esa función y pasará a regresar para imprimir los nodos inferiores y volver a imprimir los nodos superiores de otras ramas.

    Con arbolBaseB, el proceso es similar al del árbol binario, pero la cantidad de ramas que tendrá este árbol es dependiendo del valor de 'base'. Puesto que el 'base' es 3, tendrá 3 ramas y cada rama tendrá otras 3 ramas hasta que lleguen a nivel 4 y termine la función; esto de tener 3 ramas en cada nueva rama se obtiene por medio de un ciclo con k=0, mientras k sea menor al valor de 'base', k aumenta en 1, e invoca la función arbolBaseB cuyos parámetros serían que se añade un nivel más, el nodo tendrá valor de base*nodo+k (para indicar la rama), y el límite y la base se mantiene. Mediante la recursividad de esta función con el bucle, se logrará tener un árbol de 3 ramas con 4 niveles (incluyendo el nivel 0 y exceptuando el nivel 4).

20. La función ordenarRapidamente tiene de parámetro un arreglo de enteros *arr, y dos variables de tipo entero, inf (posición inferior, que inicialmente puede ser 0) y sup (posición superior, que inicialmente puede ser la longitud del arreglo -1). En esta función, se crea una variable de tipo entero llamado 'pivote' siendo 0; se comprueba si el arreglo tiene valor nulo, si es así, acaba la función. En otro caso, se verifica si inf es menor que

sup, si es el caso, la variable pivote se le asigna el valor devuelto por la función 'particionar', que se le pasa como parámetro los mismos que esta función. Luego, se invoca la función ordenarRapidamente, pasando como parámetro el arreglo arr, inf y pivote – 1, después, se invoca de nuevo la función, pero esta vez en lugar de inf es pivote+1 y, en lugar de pivote – 1, es sup.

Cuando se invoca 'particionar', se declaran las variables pivote, aux, i y j, todos con valor de 0. 'pivote' obtiene el contenido de la posición del valor de sup del arreglo (el valor de la última casilla), luego i tiene el valor de inf; después, se declara un ciclo con j teniendo el valor de inf, mientras j sea menor a sup, j aumenta en 1, en este ciclo se comprueba si la celda j del arreglo es menor al valor de 'pivote', si es así, la variable aux adquiere el contenido de la casilla j del arreglo, luego, la celda j del arreglo cambia de valor por el de la celda i del mismo arreglo, tras eso, la celda i obtiene el valor de la variable aux e i aumenta en 1; esto fue un intercambio de posiciones; si la condición no se cumple, se bucle continúa repitiendo la condición. Después del ciclo, el arreglo ya tendría que estar ordenado, y se procede a que la variable aux tenga el contenido de la posición del valor de sup del arreglo, luego, la posición del valor de sup del arreglo adquiere el valor de la casilla i del arreglo, después, la celda i del arreglo tiene el valor de la variable auxiliar. Finalmente, la función retorna el valor de i. La primera invocación de esta función no ordena el arreglo por completo, se irá completando el ordenamiento conforme las funciones recursivas vayan invocando esta misma función con ciertos valores disintos.

Cuando se invoca ordenarRapidamente por primera vez dentro de su propia función, se tiene que sup tiene el valor de pivote-1 y, a su vez, tiene valor de i-1; la invocación de esta función es para seguir ordenando el arreglo hasta que la condición de inf menor a sup se deje de cumplir debido a que sup será menor a inf. Cuando suceda eso, se empezará a invocar ordenarRapidamente por segunda ocasión dentro de la condición; aquí, como será la variable inf la que aumentará, la condición llegará a dejar de cumplirse hasta cierto punto (cuando inf sea mayor a sup), pero mientras aún no suceda eso, se terminará de ordenar el arreglo. En cuestión de la función que se invoca antes de ésta, la condición de inf menor a sup dejará de cumplirse debido a que inf se irá sumando en 1 y será mayor a sup.

De esta forma, si es que inf era 0 y sup la longitud del arreglo -1, el arreglo termina ordenado de menor a mayor.