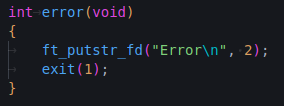


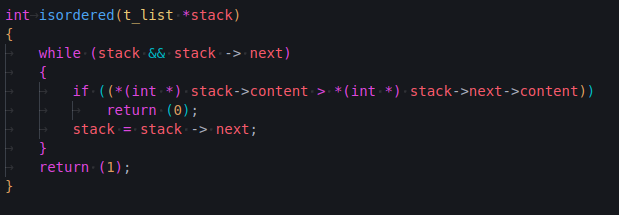
La función free\_resources está diseñada para liberar memoria dinámica utilizada en el programa. Recibe tres argumentos: dos punteros a listas enlazadas (stack\_a y stack\_b) y un array de cadenas de caracteres (splited).

Primero, comprueba si stack\_a no es nulo y, si es así, llama a ft\_lstclear para liberar todos los nodos de la lista y su contenido. Hace lo mismo con stack\_b. Después, si splited no es nulo, recorre el array liberando cada cadena individualmente y, al final, libera el propio array.

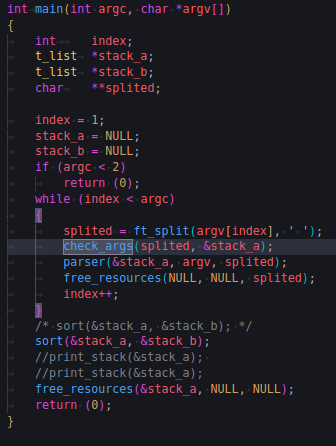
Esta función ayuda a evitar fugas de memoria asegurándose de que todos los recursos dinámicos se liberen correctamente cuando ya no se necesitan.



Sirve para imrprimit un mensaje de error.



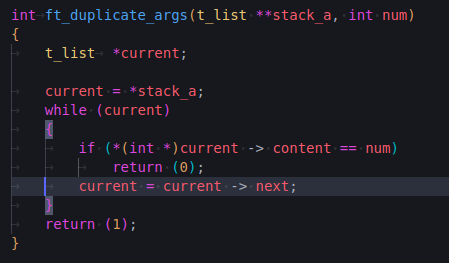
Sirve para comprobar si el stack ya esta ordenado.



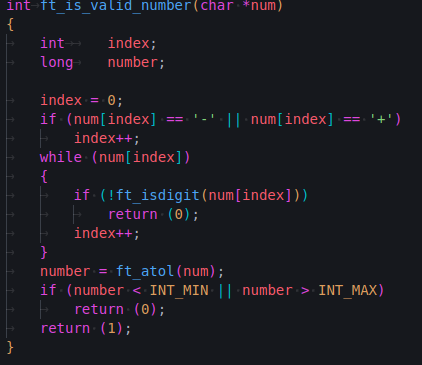
Mientras index sea menor que el numero de argumentos se splitean los argumentos, se le pasa a la funcios check\_args y si va bien lo parseamos y se libera memoria de el split

Despues llamamos a la funcion sort para ordenar los numeros que hemos obtenido y leberamos stack\_a a continuacions se explican las funciones.

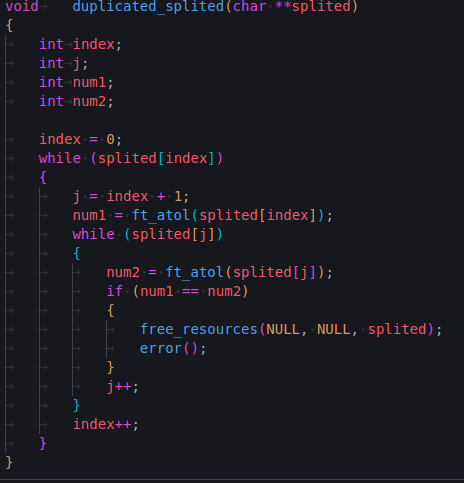
**CHECK\_ARGS**



Verifica que no haya ningun argumento repetido si lo hay devuelve 0 si no devuelve 1.



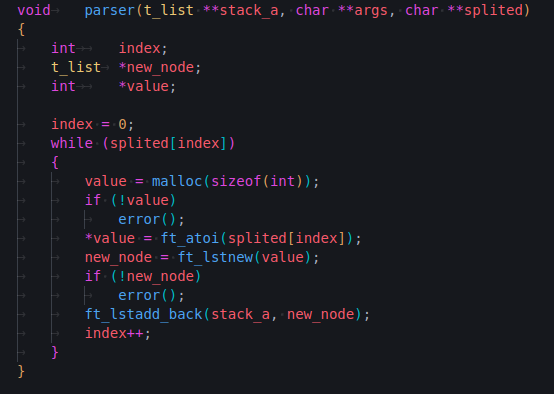
Verifica si el numero esta dentro de los limites de un int si lo esta devulve 1 si no 0.



Verifica que no haya duplicados en el split.

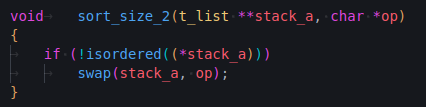


Check\_args hace todas las comprobaciones anteriores si da algun error libera y muestra el mensaje de error.

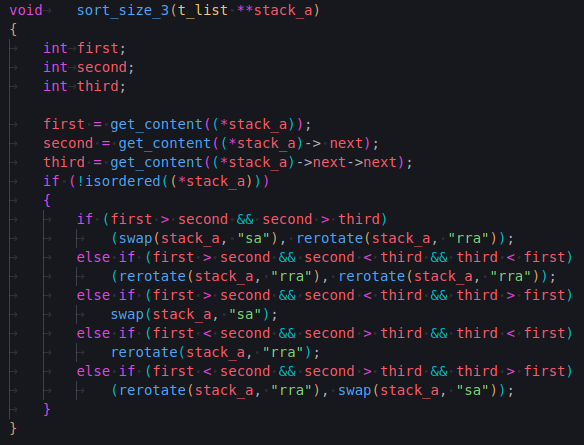


Convierte los argumentos es numeros y los almacena al final de la lista.

**SORT**

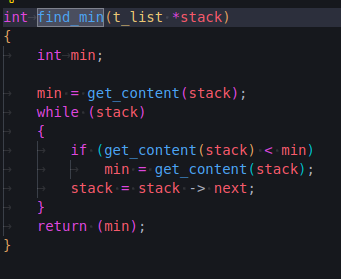
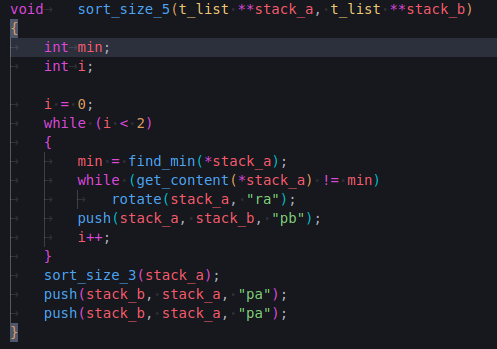


Si la lista no esta ordenada aplica la operacion swap en el stack\_a.



Coge el primer segundo y tercer valor si esta desordenado.

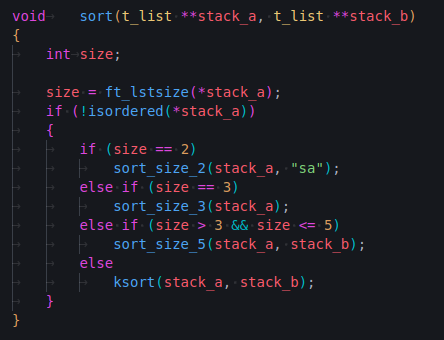
* Si first es mayor que second y second mayor que third hace sa y rra.
* Si first mayor que second , sencond mayor que third y third mayor que first hace 2 rra.
* Si first es menor que second , second mayor que third y third menor que first hace rra
* Si first es menor que second , second es mayor que third y thir es mayor que first hace rra y sa



Llama a la funcion find\_min que busca el valor mas pequeño miemtrar el contenido de arriba no sea el menor hace ra y cuando el minimo esta arriba hace pb asi por 2 veces

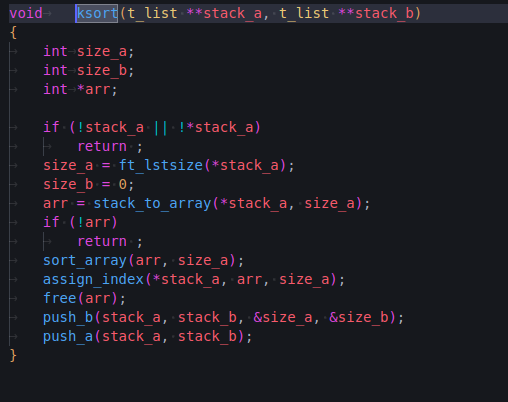
ordena stack\_a con sort\_size\_3 y si no sor\_size\_2

Por ultimo despues de ordenar stack\_a hacemos pa para devolver los 2 valores mas pequeños y asi quedara ordenado.

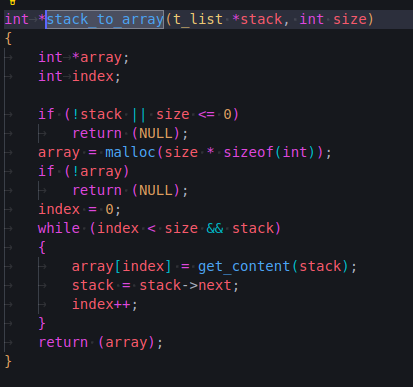


Dependiendo del tamaño de la lista llama a una funcion o a otra para ordenarla si es mayor de 5 llama a ksort.

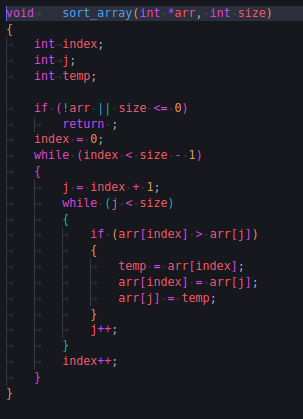
**KSORT**



Llama a stack\_to\_array sort\_arrya , assign\_index y push\_b , push\_a para ordenar la lista a



Alamcena los valores de la lista en un array;



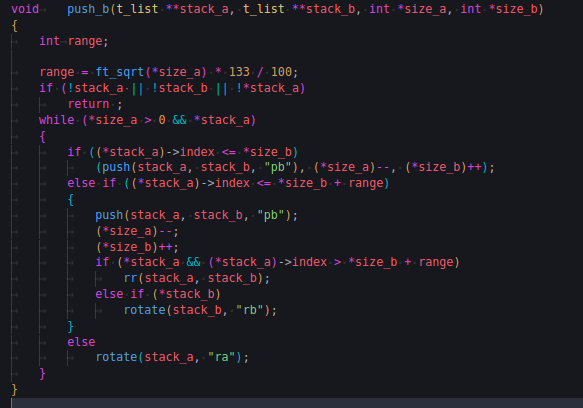
Ordena mediante el metodo burbuja de menor a mayor el array



La función assign\_index asigna un índice a cada nodo de una lista enlazada (stack) según la posición de su valor en un array ordenado (sorted\_arr). Primero, verifica que los parámetros sean válidos y reserva memoria para un array auxiliar (assigned) que sirve para marcar qué posiciones del array ordenado ya han sido asignadas.

Luego, recorre cada nodo de la lista. Para cada nodo, busca en el array ordenado la posición que coincide con el valor del nodo y que aún no ha sido asignada. Cuando encuentra esa posición, asigna ese índice al campo index del nodo y marca la posición como asignada en el array auxiliar, evitando así duplicados en caso de valores repetidos.

Finalmente, libera la memoria reservada para el array auxiliar. Esta función es útil para identificar rápidamente la posición ordenada de cada elemento de la lista, lo que facilita la implementación de algoritmos de ordenación más eficientes.



La función push\_b se encarga de mover elementos desde la pila stack\_a hacia la pila stack\_b siguiendo una estrategia basada en rangos de índices, optimizando así el proceso de ordenación en el algoritmo "push\_swap".

Primero, calcula un rango (range) que depende de la raíz cuadrada del tamaño de stack\_a, ajustado por un factor (1.33). Este rango determina cuántos elementos se consideran en cada lote para ser enviados a stack\_b. Si los punteros de las pilas no son válidos o stack\_a está vacía, la función termina inmediatamente.

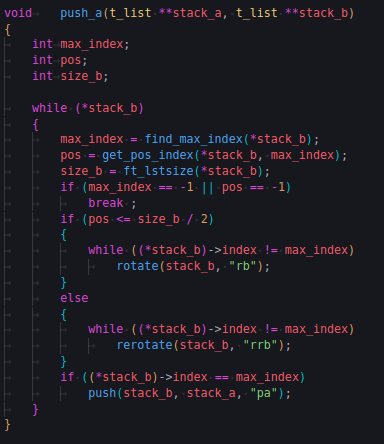
Dentro del bucle principal, mientras queden elementos en stack\_a, la función evalúa el índice del elemento superior:

Si el índice es menor o igual al tamaño actual de stack\_b, lo mueve directamente a stack\_b y actualiza los tamaños.

Si el índice está dentro del rango calculado, también lo mueve a stack\_b, actualiza los tamaños y, dependiendo del siguiente elemento, realiza una rotación doble (rr) o solo rota stack\_b para optimizar el orden.

Si el índice está fuera del rango, simplemente rota stack\_a para traer otro elemento al tope.

Esta estrategia de dividir en rangos y rotar permite distribuir los elementos en stack\_b de forma que el proceso de ordenación posterior sea más eficiente, reduciendo el número total de operaciones necesarias.



La función push\_a se encarga de mover los elementos de la pila stack\_b de vuelta a la pila stack\_a, asegurando que se haga en orden descendente según el índice de cada elemento. Este proceso es típico en algoritmos de ordenación con dos pilas, como en el proyecto "push\_swap".

Dentro de un bucle que se ejecuta mientras stack\_b no esté vacía, la función busca el índice máximo presente en stack\_b usando find\_max\_index y obtiene su posición con get\_pos\_index. También calcula el tamaño actual de stack\_b. Si no encuentra un índice válido o una posición válida, sale del bucle por seguridad.

Luego, decide la forma más eficiente de llevar el elemento con el índice máximo hasta la cima de stack\_b:

Si la posición del elemento está en la primera mitad de la pila, rota hacia arriba (rotate, "rb") hasta que el elemento esté arriba.

Si está en la segunda mitad, rota hacia abajo (rerotate, "rrb") hasta que llegue a la cima.

Una vez que el elemento con el índice máximo está en la cima de stack\_b, lo mueve a stack\_a usando la función push. Este proceso se repite hasta que stack\_b queda vacía, logrando así que los elementos se reinsertan en stack\_a en orden descendente, completando la ordenación.