INTRODUCCIÓN

Para el proyecto push_swap, se nos da una pila de números enteros y debemos ordenarla utilizando solo dos pilas (A y B) y los siguientes movimientos:

sa: intercambia los dos primeros elementos de la pila A.
sb: intercambia los dos primeros elementos de la pila B.
ss: intercambia los dos primeros elementos de ambas pilas A y B.

pa: mueve el primer elemento de la pila B a la pila A.
pb: mueve el primer elemento de la pila A a la pila B.

ra: rotación hacia la derecha de la pila A.
rb: rotación hacia la derecha de la pila B.
rr: rotación hacia la derecha de ambas pilas A y B.

rra: rotación hacia la izquierda de la pila A.
rrb: rotación hacia la izquierda de la pila B.
rrr: rotación hacia la izquierda de ambas pilas A y B.

LOS MOVIMIENTOS EN EL CÓDIGO

t_push.c

Este código define dos funciones, ‘pa’ y ‘pb’, que son utilizadas para manipular dos pilas, stack_a y stack_b. Estas funciones son parte del algoritmo de ordenación, que utiliza pilas para ordenar elementos.

La **función ‘pa’** toma tres parámetros:

- stack_a: una pila que se utiliza como destino
- stack_b: una pila que se utiliza como origen
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función hace lo siguiente:

- Verifica si la pila stack_b está vacía. Si lo está, la función devuelve sin hacer nada.
- Crea un puntero temporal tmp que apunta al segundo elemento de la pila stack_b.
- Asigna el primer elemento de la pila stack_b como el nuevo primer elemento de la pila stack_a.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de la pila stack_b.
- Si el parámetro print es verdadero, imprime el mensaje "pa\n" en la consola.

Supongamos que las pilas stack_a y stack_b tienen los siguientes elementos:

stack_a: 1 -> 2 -> 3 stack_b: 4 -> 5 -> 6

- Llamamos a la función pa con los parámetros stack_a, stack_b y 1 (para imprimir el mensaje).
- La función pa hace lo siguiente:
- Verifica que stack_b no está vacía.
- Crea un puntero temporal tmp que apunta al segundo elemento de stack_b, que es el elemento 5.
- Asigna el primer elemento de stack_b, que es el elemento 4, como el nuevo primer elemento de stack_a.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de stack_b.
- Imprime el mensaje "pa\n" en la consola.

Las pilas quedan así:
stack_a: 4 -> 1 -> 2 -> 3 stack_b: 5 -> 6

La '**función pb'** es similar a la función 'pa', pero con las pilas invertidas. La función pb toma los mismos parámetros que la función pa, pero hace lo siguiente:

- Verifica si la pila stack_a está vacía. Si lo está, la función devuelve sin hacer nada.
- Crea un puntero temporal tmp que apunta al segundo elemento de la pila stack_a.
- Asigna el primer elemento de la pila stack_a como el nuevo primer elemento de la pila stack_b.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de la pila stack_a.
- Si el parámetro print es verdadero, imprime el mensaje "pb\n" en la consola.

Supongamos que las pilas stack_a y stack_b tienen los siguientes elementos:

stack_a: 1 -> 2 -> 3 stack_b: 4 -> 5 -> 6

- Llamamos a la función pb con los parámetros stack_a, stack_b y 1 (para imprimir el mensaje).
- Verifica que stack_a no está vacía.
- Crea un puntero temporal tmp que apunta al segundo elemento de stack_a, que es el elemento 2.
- Asigna el primer elemento de stack_a, que es el elemento 1, como el nuevo primer elemento de stack_b.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de stack_a.
- Imprime el mensaje "pb\n" en la consola.

Las pilas quedan así:
stack_a: 2 -> 3 stack_b: 1 -> 4 -> 5 -> 6

ft_rotate_down

Este código define varias funciones que manipulan dos pilas, stack_a y stack_b, y realizan operaciones de rotación y movimiento de elementos entre ellas.

La **función rra** toma dos parámetros:

- stack_a: una pila que se utiliza como destino
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función rra hace lo siguiente:

- Verifica si la pila stack_a está vacía o solo tiene un elemento. Si es así, la función devuelve sin hacer nada.
- Busca el último elemento de la pila stack_a utilizando un puntero last.
- Crea un puntero temporal tmp que apunta al último elemento de la pila stack_a.
- Asigna el último elemento de la pila stack_a como el nuevo primer elemento de la pila stack_a.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de la pila stack_a.
- Si el parámetro print es verdadero, imprime el mensaje "rra\n" en la consola.

Supongamos que la pila stack_a tiene los siguientes elementos:
stack_a: 1 -> 2 -> 3 -> 4 -> 5

- Llamamos a la función rra con los parámetros stack_a y 1 (para imprimir el mensaje).
- Se verifica que stack_a no está vacía.
- Busca el último elemento de la pila stack_a, que es el elemento 5.
- Crea un puntero temporal tmp que apunta al elemento 5.
- Asigna el elemento 5 como el nuevo primer elemento de la pila stack_a.
- Asigna el puntero temporal tmp como el nuevo segundo elemento de la pila stack_a.
- Imprime el mensaje "rra\n" en la consola.
- La pila stack_a queda así:
- stack_a: 5 -> 1 -> 2 -> 3 -> 4

La **función rrb** es similar a la función rra, pero opera sobre la pila stack_b.

La **función rrr** toma tres parámetros:

- stack_a: una pila que se utiliza como destino
- stack_b: una pila que se utiliza como origen
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función rrr hace lo siguiente:

- Llama a la función rra con los parámetros stack_a y 0 (no imprime mensaje).
- Llama a la función rrb con los parámetros stack_b y 0 (no imprime mensaje).
- Si el parámetro print es verdadero, imprime el mensaje "rrr\n" en la consola.

La función rrr_rot toma cuatro parámetros:

- s_a: una pila que se utiliza como destino
- s_b: una pila que se utiliza como origen
- a_cost: un puntero a un entero que indica el costo de la operación en la pila s_a
- b_cost: un puntero a un entero que indica el costo de la operación en la pila s_b

La función rrr_rot hace lo siguiente:

- Mientras el costo en ambas pilas sea negativo, llama a la función rrr con los parámetros s_a, s_b y 1 (imprime mensaje).
- Incrementa el costo en ambas pilas en 1.

La **función b_rot** toma dos parámetros:

- stack_b: una pila que se utiliza como destino
- cost: un puntero a un entero que indica el costo de la operación en la pila stack_b

La función b_rot hace lo siguiente:

- Mientras el costo sea diferente de 0, llama a la función rb o rrb dependiendo del signo del costo.
- Si el costo es positivo, llama a la función rb con los parámetros stack_b y 1 (imprime mensaje).
- Si el costo es negativo, llama a la función `rrb

ft_rotate_up

Este código define varias funciones que manipulan dos pilas, y realizan operaciones de rotación y movimiento de elementos entre ellas.

La **función ra** toma dos parámetros:

- stack_a: una pila que se utiliza como destino
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función ra hace lo siguiente:

- Verifica si la pila stack_a está vacía o solo tiene un elemento. Si es así, la función devuelve sin hacer nada.
- Crea un puntero temporal tmp que apunta al primer elemento de la pila stack_a.
- Asigna el segundo elemento de la pila stack_a como el nuevo primer elemento de la pila stack_a.
- Busca el último elemento de la pila stack_a utilizando un puntero last.
- Asigna el puntero temporal tmp como el nuevo último elemento de la pila stack_a.
- Si el parámetro print es verdadero, imprime el mensaje "ra\n" en la consola.

Supongamos que la pila stack_a tiene los siguientes elementos:
stack_a: 1 -> 2 -> 3 -> 4 -> 5

- Llamamos a la función ra con los parámetros stack_a y 1 (para imprimir el mensaje).
- La función ra hace lo siguiente:
- Verifica que stack_a no está vacía.
- Crea un puntero temporal tmp que apunta al elemento 1.
- Asigna el elemento 2 como el nuevo primer elemento de la pila stack_a.
- Busca el último elemento de la pila stack_a, que es el elemento 5.
- Asigna el puntero temporal tmp como el nuevo último elemento de la pila stack_a.
- Imprime el mensaje "ra\n" en la consola.

La pila stack_a queda así:
stack_a: 2 -> 3 -> 4 -> 5 -> 1

La **función rb** es similar a la función ra, pero opera sobre la pila stack_b.

La **función rr** toma tres parámetros:

- stack_a: una pila que se utiliza como destino
- stack_b: una pila que se utiliza como origen
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función rr hace lo siguiente:

- Llama a la función ra con los parámetros stack_a y 0 (no imprime mensaje).
- Llama a la función rb con los parámetros stack_b y 0 (no imprime mensaje).
- Si el parámetro print es verdadero, imprime el mensaje "rr\n" en la consola.

La **función rr_rot** toma cuatro parámetros:

- s_a: una pila que se utiliza como destino
- s_b: una pila que se utiliza como origen
- a_cost: un puntero a un entero que indica el costo de la operación en la pila s_a
- b_cost: un puntero a un entero que indica el costo de la operación en la pila s_b

La función rr_rot hace lo siguiente:

- Mientras el costo en ambas pilas sea positivo, llama a la función rr con los parámetros s_a, s_b y 1 (imprime mensaje).
- Decrementa el costo en ambas pilas en 1.

La **función a_rot** toma dos parámetros:

- stack_a: una pila que se utiliza como destino
- cost: un puntero a un entero que indica el costo de la operación en la pila stack_a

La función a_rot hace lo siguiente:

- Mientras el costo sea diferente de 0, llama a la función ra o rra dependiendo del signo del costo.
- Si el costo es positivo, llama a la función ra con los parámetros stack_a y 1 (imprime mensaje).
- Si el costo es negativo, llama a la función rra con los parámetros stack_a y 1 (imprime mensaje).

ft_swap.c

Este código define tres funciones que manipulan dos pilas, stack_a y stack_b, y realizan operaciones de swap (intercambio) entre los elementos de las pilas.

La **función sa** toma dos parámetros:

- stack_a: una pila que se utiliza como destino
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función sa hace lo siguiente:

- Verifica si la pila stack_a está vacía o solo tiene un elemento. Si es así, la función devuelve sin hacer nada.
- Crea un puntero temporal tmp que apunta al primer elemento de la pila stack_a.
- Asigna el segundo elemento de la pila stack_a como el nuevo primer elemento de la pila stack_a.
- Asigna el tercer elemento de la pila stack_a como el nuevo segundo elemento de la pila stack_a.
- Asigna el puntero temporal tmp como el nuevo tercer elemento de la pila stack_a.
- Si el parámetro print es verdadero, imprime el mensaje "sa\n" en la consola.

Supongamos que la pila stack_a tiene los siguientes elementos:
stack_a: 1 -> 2 -> 3 -> 4 -> 5

- Llamamos a la función sa con los parámetros stack_a y 1 (para imprimir el mensaje).
- La función sa hace lo siguiente:
- Verifica que stack_a no está vacía.
- Crea un puntero temporal tmp que apunta al elemento 1.
- Asigna el elemento 2 como el nuevo primer elemento de la pila stack_a.
- Asigna el elemento 3 como el nuevo segundo elemento de la pila stack_a.
- Asigna el puntero temporal tmp como el nuevo tercer elemento de la pila stack_a.
- Imprime el mensaje "sa\n" en la consola.

La pila stack_a queda así:
stack_a: 2 -> 1 -> 3 -> 4 -> 5

La **función sb** es similar a la función sa, pero opera sobre la pila stack_b.

La **función ss** toma tres parámetros:

- stack_a: una pila que se utiliza como destino
- stack_b: una pila que se utiliza como origen
- print: un booleano que indica si se debe imprimir un mensaje en la consola

La función ss hace lo siguiente:

- Llama a la función sa con los parámetros stack_a y 0 (no imprime mensaje).
- Llama a la función sb con los parámetros stack_b y 0 (no imprime mensaje).
- Si el parámetro print es verdadero, imprime el mensaje "ss\n" en la consola.

En resumen, las funciones sa y sb realizan un swap entre los dos primeros elementos de cada pila, mientras que la función ss llama a ambas funciones y realiza un swap simultáneo en ambas pilas.

ft_tindex.c

Durante la mayor parte del programa, este no utiliza los valores que se les han introducido por argumentos, si no que se usan los índices de cada uno de ellos. Esto resulta más cómodo y práctico para el desarrollo del código. Las siguientes funciones son las encargadas de generar estos índices (enteros ordenados).

La **función copy_stack** toma un puntero a una pila original como parámetro y devuelve una copia de esa pila.

- Se crea un puntero copy que apunta a la cabeza de la pila copia.
- Se crea un puntero last que apunta al último elemento de la pila copia.
- Se itera sobre la pila original utilizando un puntero original.
- En cada iteración, se crea un nuevo elemento temp en la pila copia utilizando malloc.
- Se asigna el valor del elemento original al elemento copia.
- Se enlaza el elemento copia con el elemento anterior en la pila copia utilizando last.
- Se actualiza el puntero last para apuntar al nuevo elemento copia.
- Se devuelve la cabeza de la pila copia.

  Supongamos que la pila original es:
  original: 1 -> 2 -> 3 -> 4 -> 5

  Llamamos a la función copy_stack con la pila original como parámetro.

  La función copy_stack devuelve una copia de la pila original:
  copy: 1 -> 2 -> 3 -> 4 -> 5

La **función sort_list** toma un puntero a una pila list como parámetro y ordena la pila en orden ascendente.

- Se itera sobre la pila utilizando un puntero temp.
- En cada iteración, se itera sobre la pila utilizando un puntero temp2 que apunta al elemento siguiente.
- Se compara el valor del elemento actual con el valor del elemento siguiente.
- Si el valor del elemento actual es mayor que el valor del elemento siguiente, se intercambian los valores.
- Se repite el proceso hasta que la pila esté ordenada.

  Supongamos que la pila original es:
  list: 5 -> 2 -> 8 -> 3 -> 1

- Llamamos a la función sort_list con la pila original como parámetro.
- La función sort_list ordena la pila en orden ascendente:
- list: 1 -> 2 -> 3 -> 5 -> 8

La **función find_index** toma un puntero a una pila sorted_list y un valor value como parámetros y devuelve el índice del valor en la pila ordenada.

- Se itera sobre la pila ordenada utilizando un puntero sorted_list.
- Se compara el valor del elemento actual con el valor buscado.

- Si se encuentra el valor, se devuelve el índice actual.
- Si no se encuentra el valor, se devuelve -1.

  Supongamos que la pila ordenada es:
  sorted_list: 1 -> 2 -> 3 -> 5 -> 8
  Llamamos a la función find_index con la pila ordenada y el valor 3 como parámetros.
  La función find_index devuelve el índice 3.

La función **ft_get_index** toma un puntero a una pila stack_a como parámetro y asigna un índice a cada elemento de la pila.

- Se crea una copia de la pila original utilizando la función copy_stack.
- Se ordena la copia de la pila utilizando la función sort_list.
- Se itera sobre la pila original utilizando un puntero original.
- Para cada elemento, se busca su índice en la pila ordenada utilizando la función find_index.
- Se asigna el índice encontrado al elemento original.
- Se libera la memoria utilizada por la copia de la pila.

  Supongamos que la pila original es:
  stack_a: 5 -> 2 -> 8 -> 3 -> -555
  Llamamos a la función ft_get_index con la pila original como parámetro.
  La función ft_get_index asigna un índice a cada elemento de la pila:
  stack_a: 5 (índice 4), 2 (índice 2), 8 (índice 5), 3(índice 3), -555(índice 1)

ft_find_index.c

La f**unción ft_find_index_up** toma tres parámetros:

- stack_a: un puntero a una pila
- min: un valor mínimo
- max: un valor máximo

  La función devuelve el índice del primer elemento en la pila stack_a que tiene un índice entre min y max (inclusive).

- Se crea un puntero tmp que apunta a la cabeza de la pila stack_a.
- Se crea una variable index que se inicializa en 1.
- Se itera sobre la pila utilizando el puntero tmp.
- En cada iteración, se verifica si el índice del elemento actual está entre min y max.
- Si se encuentra un elemento que cumple la condición, se devuelve el índice actual.
- Si no se encuentra ningún elemento que cumpla la condición, se devuelve -1.

  Supongamos que la pila stack_a es:
  stack_a: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4) -> 5 (índice 5)
  Llamamos a la función ft_find_index_up con los parámetros stack_a, 2 y 4.
  La función devuelve el índice 2, que es el índice del elemento 2 en la pila.

La función **ft_find_index_down** es similar a ft_find_index_up, pero devuelve el índice del último elemento en la pila stack_a que tiene un índice entre min y max (inclusive).

Supongamos que la pila stack_a es:
stack_a: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4) -> 5 (índice 5)
Llamamos a la función ft_find_index_down con los parámetros stack_a, 2 y 4.
La función devuelve el índice 4, que es el índice del elemento 4 en la pila.

La **función ft_cheap_sort** toma dos parámetros:

- stack_a: un puntero a una pila
- stack_b: un puntero a otra pila

La función busca el elemento en la pila stack_b que tiene el costo más bajo para ser movido a la pila stack_a y lo mueve.

- Se crea un puntero tmp que apunta a la cabeza de la pila stack_b.
- Se crea una variable cheap que se inicializa en INT_MAX.
- Se itera sobre la pila stack_b utilizando el puntero tmp.
- En cada iteración, se calcula el costo actual del elemento utilizando las variables a_cost y b_cost.
- Si el costo actual es menor que el costo más bajo encontrado hasta ahora, se actualiza el costo más bajo y se guardan los valores de a_cost y b_cost.
- Una vez que se ha encontrado el elemento con el costo más bajo, se llama a la función ft_moves para mover el elemento a la pila stack_a.

Supongamos que la pila stack_b es:
stack_b: 1 (a_cost = 2, b_cost = 1) -> 2 (a_cost = 3, b_cost = 2) -> 3 (a_cost = 1, b_cost = 3)
Llamamos a la función ft_cheap_sort con los parámetros stack_a y stack_b.
La función devuelve el elemento 3, que tiene el costo más bajo para ser movido a la pila stack_a.

La **función ft_get_cost** toma dos parámetros:

- stack_a: un puntero a una pila
- stack_b: un puntero a otra pila

La función calcula el costo de cada elemento en la pila stack_b para ser movido a la pila stack_a.

- Se crea un puntero tmp_b que apunta a la cabeza de la pila stack_b.
- Se calculan las longitudes de las pilas stack_a y stack_b utilizando las funciones ft_stack_len.
- Se itera sobre la pila stack_b utilizando el puntero tmp_b.
- En cada iteración, se calcula el costo b_cost del elemento actual como su posición en la pila stack_b. Si la posición es mayor que la mitad de la longitud de la pila, se calcula como la diferencia entre la longitud de la pila y la posición.
- Se calcula el costo a_cost del elemento actual como su posición objetivo en la pila stack_a. Si la posición objetivo es mayor que la mitad de la longitud de la pila, se calcula como la diferencia entre la longitud de la pila y la posición objetivo.

- Se asignan los costos calculados a los campos b_cost y a_cost del elemento actual.

  Supongamos que la pila stack_b es:
  stack_b: 1 (pos = 0) -> 2 (pos = 1) -> 3 (pos = 2)
  Y la pila stack_a tiene una longitud de 5.
  Llamamos a la función ft_get_cost con los parámetros stack_a y stack_b.
  La función calcula los costos de cada elemento en la pila stack_b como sigue:
  Elemento 1: b_cost = 0, a_cost = 2 (ya que su posición objetivo es 2 en la pila stack_a)
  Elemento 2: b_cost = 1, a_cost = 3 (ya que su posición objetivo es 3 en la pila stack_a)
  Elemento 3: b_cost = 2, a_cost = 1 (ya que su posición objetivo es 1 en la pila stack_a)

ft_tfind.c

La **función ft_find_min** toma un parámetro:

- stack: un puntero a una pila

  La función devuelve el índice mínimo de los elementos en la pila stack.

- Se crea un puntero tmp que apunta a la cabeza de la pila stack.
- Se crea una variable min que se inicializa con el índice del primer elemento de la pila.
- Se itera sobre la pila utilizando el puntero tmp.
- En cada iteración, se verifica si el índice del elemento actual es menor que el valor actual de min.
- Si se encuentra un elemento con un índice menor, se actualiza el valor de min.
- Una vez que se ha iterado sobre toda la pila, se devuelve el valor de min.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
  Llamamos a la función ft_find_min con el parámetro stack.
  La función devuelve el valor 1, que es el índice mínimo de los elementos en la pila.

La función **ft_find_max** es similar a ft_find_min, pero devuelve el índice máximo de los elementos en la pila stack.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
  Llamamos a la función ft_find_max con el parámetro stack.
  La función devuelve el valor 4, que es el índice máximo de los elementos en la pila.

La función ft_find_pos_min toma dos parámetros:

- stack: un puntero a una pila
- min: el índice mínimo encontrado por la función ft_find_min

  La función devuelve la posición del elemento con el índice min en la pila stack.

- Se crea un puntero tmp que apunta a la cabeza de la pila stack.

- Se crea una variable pos_min que se inicializa en 1.
- Se itera sobre la pila utilizando el puntero tmp.
- En cada iteración, se verifica si el índice del elemento actual es igual a min.
- Si se encuentra el elemento con el índice min, se devuelve la posición actual pos_min.
- Si no se encuentra el elemento, se devuelve la posición actual pos_min.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
  Llamamos a la función ft_find_min con el parámetro stack y obtenemos el valor 1.
  Luego, llamamos a la función ft_find_pos_min con los parámetros stack y 1.
  La función devuelve el valor 2, que es la posición del elemento con el índice 1 en la pila.

La **función ft_find_pos_max** es similar a ft_find_pos_min, pero devuelve la posición del elemento con el índice máximo en la pila stack.

Supongamos que la pila stack es:
stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
Llamamos a la función ft_find_max con el parámetro stack y obtenemos el valor 4.
Luego, llamamos a la función ft_find_pos_max con los parámetros stack y 4.
La función devuelve el valor 4, que es la posición del elemento con el índice 4 en la pila.

La **función ft_find_last** toma un parámetro:

- stack: un puntero a una pila

  La función devuelve el índice del último elemento de la pila stack.

- Se crea un puntero tmp que apunta a la cabeza de la pila stack.
- Se itera sobre la pila utilizando el puntero tmp hasta que se llega al final de la pila.
- Se devuelve el índice del último elemento de la pila.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
  Llamamos a la función ft_find_last con el parámetro stack.
  La función devuelve el valor 4, que es el índice del último elemento de la pila.

Nota importante:

En las funciones ft_find_min, ft_find_max, ft_find_pos_min, ft_find_pos_max y ft_find_last, se llama a la función ft_free con el parámetro &tmp al final de cada función. Esto puede que sea innecesario, ya que ft_free se utiliza para liberar memoria dinámica, pero en este caso se está pasando un puntero a una variable local tmp. Esto, puede no tener mucho sentido y, aunque no ha sido el caso, en absoluto, en otras circunstancias quizás podría causar problemas de memoria. Sería valorable, por innecesaria tal vez, eliminar la llamada a ft_free en cada función.

ft_tfree.c

Este código define dos funciones que se encargan de liberar memoria dinámica utilizada por pilas y listas enlazadas.

La **función ft_free** toma un parámetro:

- stack: un puntero a un puntero a una pila (un doble puntero)

  La función se encarga de liberar toda la memoria dinámica utilizada por la pila stack.

- Se verifica si la pila stack es nula. Si es así, se devuelve inmediatamente, ya que no hay memoria que liberar.
- Se crea un puntero tmp que se utiliza para almacenar el siguiente elemento de la pila.
- Se itera sobre la pila utilizando el puntero tmp. En cada iteración:
- Se almacena el siguiente elemento de la pila en tmp.
- Se libera la memoria dinámica utilizada por el elemento actual de la pila utilizando free(*stack).
- Se actualiza el puntero *stack para que apunte al siguiente elemento de la pila (tmp).
- Una vez que se ha iterado sobre toda la pila, se establece *stack en NULL para indicar que la pila está vacía.
- Finalmente, se libera la memoria dinámica utilizada por el último elemento de la pila (tmp) utilizando free(tmp).

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)

- Llamamos a la función ft_free con el parámetro &stack.
- La función itera sobre la pila, liberando la memoria dinámica utilizada por cada elemento:
- Primero, se libera la memoria dinámica utilizada por el elemento 3.
- Luego, se libera la memoria dinámica utilizada por el elemento 1.
- Después, se libera la memoria dinámica utilizada por el elemento 2.
- Finalmente, se libera la memoria dinámica utilizada por el elemento 4.
- Una vez que se ha liberado toda la memoria dinámica, se establece *stack en NULL para indicar que la pila está vacía.

La **función ft_free_all** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de liberar toda la memoria dinámica utilizada por ambas pilas stack_a y stack_b.

- Se llama a la función ft_free con el parámetro stack_a para liberar la memoria dinámica utilizada por la pila A.
- Se llama a la función ft_free con el parámetro stack_b para liberar la memoria dinámica utilizada por la pila B.

  Supongamos que las pilas stack_a y stack_b son:
  stack_a: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)

stack_b: 4 (índice 4) -> 5 (índice 5) -> 6 (índice 6)

- Llamamos a la función ft_free_all con los parámetros &stack_a y &stack_b.
- La función llama a ft_free con stack_a y libera la memoria dinámica utilizada por la pila A.
- Luego, llama a ft_free con stack_b y libera la memoria dinámica utilizada por la pila B.
- Una vez que se ha liberado toda la memoria dinámica, se establecen *stack_a y *stack_b en NULL para indicar que ambas pilas están vacías.

ft_tsort.c

Este código define varias funciones que se encargan de ordenar una pila de enteros utilizando algoritmos de ordenamiento específicos.

La **función ft_stack_len** toma un parámetro:

- stack: un puntero a una pila

  La función devuelve la longitud de la pila stack.

- Se crea un puntero tmp que apunta a la cabeza de la pila stack.
- Se inicializa una variable stack_len en 0.
- Se itera sobre la pila utilizando el puntero tmp. En cada iteración, se incrementa stack_len en 1.
- Se devuelve stack_len al final de la función.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)

- Llamamos a la función ft_stack_len con el parámetro stack.
- La función devuelve 4, que es la longitud de la pila.

La **función ft_sort** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de ordenar la pila stack_a utilizando diferentes algoritmos de ordenamiento dependiendo de la longitud de la pila.

- Se verifica si la pila stack_a es nula o vacía. Si es así, se devuelve inmediatamente.
- Se llama a la función ft_stack_len para obtener la longitud de la pila stack_a.
- Se llama a la función ft_get_index para asignar índices a los elementos de la pila stack_a.
- Se llama a la función ft_add_pos para agregar posiciones a los elementos de la pila stack_a.
- Se llama a la función ft_find_pos_target para encontrar la posición objetivo en la pila stack_a.
- Se verifica la longitud de la pila stack_a y se llama a la función de ordenamiento correspondiente:
- Si la longitud es menor que 2, se devuelve inmediatamente.
- Si la longitud es 2, se llama a la función sa para ordenar la pila.

- Si la longitud es 3, se llama a la función ft_sort_three para ordenar la pila.
- Si la longitud es 4, se llama a la función ft_sort_four para ordenar la pila.
- Si la longitud es 5, se llama a la función ft_sort_five para ordenar la pila.
- Si la longitud es mayor que 5, se llama a la función ft_sort_max para ordenar la pila.

La **función ft_sort_three** toma un parámetro:

- stack: un puntero a un puntero a una pila

  La función se encarga de ordenar una pila de 3 elementos utilizando operaciones de rotación y swap.

  Supongamos que la pila stack es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)
  Llamamos a la función ft_sort_three con el parámetro stack.
  La función ordena la pila utilizando operaciones de rotación y swap.

  Este algoritmo se explica detalladamente, un poco más adelante.

La **función ft_sort_four** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de ordenar una pila de 4 elementos utilizando operaciones de rotación, swap y push.

- Se verifica la posición del elemento mínimo en la pila stack_a y se aplica la operación de rotación correspondiente para llevarlo a la posición correcta.
- Se llama a la función pb para push el elemento mínimo a la pila stack_b.
- Se verifica si la pila stack_a está ordenada. Si es así, se llama a la función pa para pop el elemento mínimo de la pila stack_b y pushlo a la pila stack_a.
- Si la pila stack_a no está ordenada, se llama a la función ft_sort_three para ordenar la pila stack_a y luego se llama a la función pa para pop el elemento mínimo de la pila stack_b y pushlo a la pila stack_a.

  Supongamos que la pila stack_a es:
  stack_a: 4 (índice 4) -> 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)
  Llamamos a la función ft_sort_four con los parámetros stack_a y stack_b.
  La función ordena la pila stack_a utilizando operaciones de rotación, swap y push.

  Este algoritmo se explica detalladamente, un poco más adelante.

La **función ft_sort_five** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

La función se encarga de ordenar una pila de 5 elementos utilizando operaciones de rotación, swap y push.

- Se verifica la posición del elemento mínimo en la pila stack_a y se aplica la operación de rotación correspondiente para llevarlo a la posición correcta.
- Se llama a la función pb para push el elemento mínimo a la pila stack_b.
- Se repite el proceso hasta que la pila stack_a tenga solo 3 elementos.
- Se llama a la función ft_sort_three para ordenar la pila stack_a.
- Se llama a la función pa para pop los elementos de la pila stack_b y pushlos a la pila stack_a.

Supongamos que la pila stack_a es:
stack_a: 5 (índice 5) -> 4 (índice 4) -> 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)
Llamamos a la función ft_sort_five con los parámetros stack_a y stack_b.
La función ordena la pila stack_a utilizando operaciones de rotación, swap y push.

Este algoritmo se explica detalladamente, un poco más adelante.

En resumen, este código define varias funciones que se encargan de ordenar pilas de enteros utilizando algoritmos de ordenamiento específicos.

Las funciones ft_sort_three, ft_sort_four y ft_sort_five se encargan de ordenar pilas de 3, 4 y 5 elementos, respectivamente, utilizando operaciones de rotación, swap y push.

PRIMEROS PASOS: ORDENAR DOS NÚMEROS

La lógica de ordenar dos números es obvia. Sí introducidos los números, el contenido des stack_a no está ordenado usamos `sa`, por ejemplo, y ya tenemos la pila ordenada.

ORDENAR TRES NÚMEROS

Hay cinco casos posibles para que se coloquen solo tres números aleatorios en la pila A. El objetivo es asegurarse de ordenarlos del más pequeño al más grande en no más de dos acciones. La forma en que se determina qué acciones debe usarse, depende de la posición del número superior, el número del medio y el número inferior. En cada caso, se compara el superior con el medio, el medio con el inferior y el inferior con el superior. Dependiendo de qué número sea mayor o menor, afectará las acciones que se invocan.

Caso 1		
2		1
1	sa	2
3		3

Caso 2				
3		2		1
2	sa	3	rr	2
1		1		3

Caso 3		
3		1
1	ra	2
2		3

Caso 4				
1		3		1
3	sa	1	ra	2
2		2		3

Caso 5		
2		1
3	rra	2
1		3

void ft_sort_three(t_stack **stack)

La función `ft_sort_three` ordena una pila de tres elementos utilizando un conjunto de operaciones específicas. La pila está representada por una estructura de datos `t_stack`, y las operaciones disponibles son `sa`, `ra`, y `rra`.

Descripción de la Función

1. Entrada: `t_stack **stack` : Un puntero a un puntero a la pila que se va a ordenar.

2. Operaciones Disponibles:
- `sa(stack, 1)` : Intercambia los dos primeros elementos de la pila.
- `ra(stack, 1)` : Rota la pila hacia arriba (el primer elemento se mueve al final).
- `rra(stack, 1)` : Rota la pila hacia abajo (el último elemento se mueve al frente).

Lógica de la Función

La función utiliza una serie de condiciones `if-else` para determinar el orden actual de los tres elementos y aplicar las operaciones necesarias para ordenarlos.

Caso 1: Si el primer elemento es mayor que el segundo pero menor que el tercero: se intercambian los dos primeros elementos (`sa`).

Caso 2: Si el primer elemento es mayor que ambos, y el segundo es mayor que el tercero: se intercambian los dos primeros elementos (`sa`), y se rota la pila hacia abajo (`rra`).

Caso 3: Si el primer elemento es mayor que ambos, pero el segundo es menor que el tercero: se rota la pila hacia arriba (`ra`).

Caso 4: Si el primer elemento es menor que ambos: se intercambian los dos primeros elementos (`sa`), y se rota la pila hacia arriba (`ra`).

Caso 5: Si el primer elemento es menor que el segundo pero mayor que el tercero: se rota la pila hacia abajo (`rra`).

En resumen:

La función `ft_sort_three` ordena una pila de tres elementos utilizando una serie de condiciones para determinar el orden actual de los elementos y aplicar las operaciones necesarias (`sa`, `ra`, `rra`) para ordenarlos correctamente.

ORDENAR CUATRO NÚMEROS

Paso 1: Enviamos el primer número superior de la pila A, a la pila B.
Paso 2-3: Utilizamos la lógica de 3 números aleatorios para ordenar los números en A.
Paso 4-6: Se asegura de que la pila A pueda aceptar correctamente los números de la pila B.

void ft_sort_four(t_stack **stack_a, t_stack **stack_b)

La función `ft_sort_four` ordena una pila de cuatro elementos utilizando una pila auxiliar y una serie de operaciones específicas. La pila está representada por una estructura de datos `t_stack`, y las operaciones disponibles son `ra`, `rra`, `pb`, `pa`, y `ft_sort_three`.

Paso a paso de la función:

1. Entrada: `t_stack **stack_a`: Un puntero a un puntero a la pila principal que se va a ordenar.
2. `t_stack **stack_b`: Un puntero a un puntero a la pila auxiliar.
3. Encuentra y mueve el mínimo a `stack_b`: la función primero encuentra la posición del elemento mínimo en `stack_a` usando `ft_find_pos_min`. Si la posición del mínimo es menor que 2 (es decir, está en la primera mitad de la pila), rota la pila hacia arriba (`ra`) hasta que el mínimo esté en la cima. Si la posición del mínimo es mayor o igual a 2 (es decir, está en la segunda mitad de la pila), rota la pila hacia abajo (`rra`) hasta que el mínimo esté en la cima. Una vez que el mínimo está en la cima, se mueve a `stack_b` usando `pb`.
4. Ordena los Tres Elementos Restantes en `stack_a`: Si los tres elementos restantes en `stack_a` ya están ordenados (`be_sorted` devuelve 1), simplemente mueve el elemento de `stack_b` de vuelta a `stack_a` usando `pa`. Si no están ordenados, llama a `ft_sort_three` para ordenar los tres elementos restantes en `stack_a`, y luego mueve el elemento de `stack_b` de vuelta a `stack_a` usando `pa`.

Breve descripción de las Funciones Auxiliares:

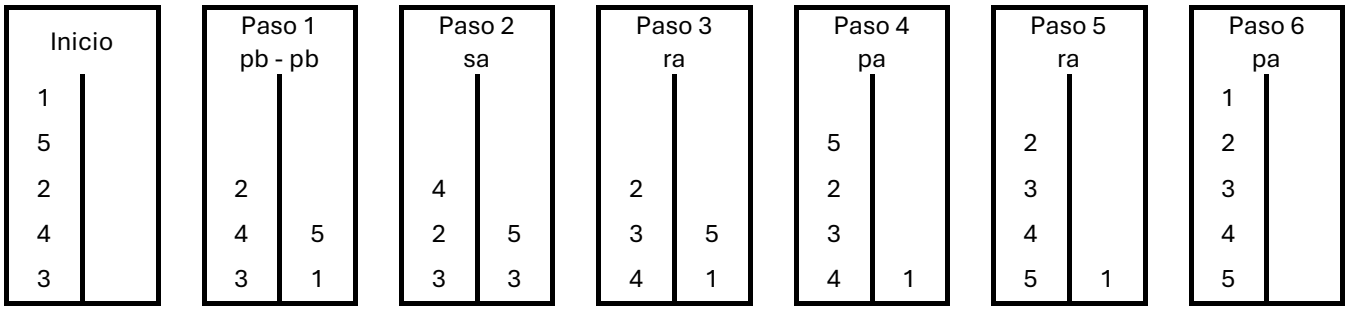
- `ft_find_pos_min(stack_a, ft_find_min(stack_a))`: Encuentra la posición del elemento mínimo en `stack_a`.
- `ft_find_min(stack_a)`: Encuentra el valor mínimo en `stack_a`.
- `ra(stack_a, 1)`: Rota `stack_a` hacia arriba.
- `rra(stack_a, 1)`: Rota `stack_a` hacia abajo.
- `pb(stack_a, stack_b, 1)`: Mueve el elemento superior de `stack_a` a `stack_b`.
- `pa(stack_a, stack_b, 1)`: Mueve el elemento superior de `stack_b` a `stack_a`.
- `be_sorted(*stack_a)`: Verifica si `stack_a` está ordenada.
- `ft_sort_three(stack_a)`: Ordena tres elementos en `stack_a`.

En resumen:

La función `ft_sort_four` ordena una pila de cuatro elementos moviendo el elemento mínimo a una pila auxiliar, ordenando los tres elementos restantes, y luego moviendo el elemento mínimo de vuelta a la pila principal. Utiliza rotaciones y operaciones de intercambio para lograr esto.

ORDENAR CINCO NÚMEROS

Trabajamos ahora con 5 números aleatorios que se colocan en la pila A. Solo se nos permiten 12 acciones, y cualquier cosa más allá de eso hará que falle esta sección del proyecto. De nuevo usaremos r la lógica de tres números aleatorios para optimizar nuestro código. Todo lo que tenemos que hacer es mover los primeros dos números de la parte superior de la pila A y moverlos a la pila B. Recuperaremos esos números una vez que los tres números de la pila A estén ordenados del más pequeño al más grande. Veamos un ejemplo de prueba con los números: 1 5 2 4 y 3.



Paso 1: Enviamos los dos números superiores de la pila A, a la pila B.
Paso 2-3: Utilizamos la lógica de 3 números aleatorios para ordenar los números en A.
Paso 4-6: Se asegura de que la pila A pueda aceptar correctamente los números de la pila B.

En total, utilizarás 7 puntos de acción, muy por debajo de nuestro límite máximo de 12.

void ft_sort_five(t_stack **stack_a, t_stack **stack_b)

La función `ft_sort_five` ordena una pila de cinco elementos utilizando una pila auxiliar y una serie de operaciones específicas. La pila está representada por una estructura de datos `t_stack`, y las operaciones disponibles son `ra`, `rra`, `pb`, `pa`, y `ft_sort_three`.

Paso a paso de la función

1. 1. Entrada: `t_stack **stack_a`: Un puntero a un puntero a la pila principal que se va a ordenar.
2. `t_stack **stack_b`: Un puntero a un puntero a la pila auxiliar.
3. Mover Elementos a `stack_b` Hasta que `stack_a` tenga tres elementos: la función entra en un bucle `while` que continúa hasta que `stack_a` tenga solo 3 elementos (`ft_stack_len(*stack_a) > 3`). Dentro del bucle, se encuentra la posición del elemento mínimo en `stack_a` usando `ft_find_pos_min`. Si la posición del mínimo es menor que 3 (es decir, está en la primera mitad de la pila), rota la pila hacia arriba (`ra`) hasta que el mínimo esté en la cima. Si la posición del mínimo es mayor o igual a 3 (es decir, está en la segunda mitad de la pila), rota la pila hacia abajo (`rra`) hasta que el mínimo esté en la cima. Una vez que el mínimo está en la cima, se mueve a `stack_b` usando `pb`.
4. Ordenar los tres elementos restantes en `stack_a`: Si los tres elementos restantes en ‘stac_a`ya están ordenados (be_sorted devuelve 1), simplemente nueve los dos elementos de ´stack_b` de vuelta a ‘stack_a` usando (`pa`) dos veces. Si no están ordenados, llama a `ft_sort_three` para ordenar los tres elementos restantes en `stack_a`, y luego mueve los dos elementos de `stack_b` de vuelta a `stack_a` usando (`pa`) dos veces.

Breve descripción de las Funciones Auxiliares

- `ft_stack_len(*stack_a)`: Devuelve la longitud de `stack_a`.
- `ft_find_pos_min(stack_a, ft_find_min(stack_a))`: Encuentra la posición del elemento mínimo en `stack_a`.
- `ft_find_min(stack_a)`: Encuentra el valor mínimo en `stack_a`.
- `ra(stack_a, 1)`: Rota `stack_a` hacia arriba.

- `rra(stack_a, 1)`: Rota `stack_a` hacia abajo.
- `pb(stack_a, stack_b, 1)`: Mueve el elemento superior de `stack_a` a `stack_b`.
- `pa(stack_a, stack_b, 1)`: Mueve el elemento superior de `stack_b` a `stack_a`.
- `be_sorted(*stack_a)`: Verifica si `stack_a` está ordenada.
- `ft_sort_three(stack_a)`: Ordena tres elementos en `stack_a`.

En resumen:

La función `ft_sort_five` ordena una pila de cinco elementos moviendo los dos elementos mínimos a una pila auxiliar, ordenando los tres elementos restantes en la pila principal, y luego moviendo los dos elementos mínimos de vuelta a la pila principal. Utiliza rotaciones y operaciones de intercambio para lograr esto.

ORDENAR n>5 NÚMEROS

Algoritmo de costes

El algoritmo de costes, el utilizado en este documento, se basa en la idea de asignar un "coste" a cada posible movimiento en función de su impacto en la pila. El objetivo, en resumen, es minimizar el coste total de los movimientos necesarios para ordenar la pila.

¿Qué es el algoritmo de costes?

El algoritmo de costes es un enfoque para resolver problemas que involucran encontrar la mejor secuencia de acciones para alcanzar un objetivo. En este caso, el objetivo es ordenar una pila de números enteros utilizando un conjunto de movimientos permitidos.

¿Cómo funciona?

Asignar costes:

Se asigna un "coste" a cada movimiento permitido. El coste es una medida de cuán "caro" es realizar ese movimiento. Por ejemplo, si un movimiento es muy beneficioso, se le asigna un coste bajo. Si un movimiento es poco beneficioso, se le asigna un coste alto.

Explorar movimientos:

Se explora todos los posibles movimientos que se pueden realizar en cada paso. Se calcula el nuevo coste total si se realiza cada movimiento.

Seleccionar el mejor movimiento:

Se selecciona el movimiento con el menor coste total. Este movimiento es el que se considera "mejor" en ese momento.

Actualizar el estado: Se actualiza el estado de la pila según el movimiento seleccionado.

Repetir: Se repite el proceso hasta que se alcanza el objetivo (en este caso, ordenar la pila) o se alcanza un límite de profundidad (un número máximo de movimientos).

Ejemplo:

Supongamos que queremos ordenar la pila [5, 2, 8, 3, 1, 4, 6] utilizando los movimientos sa, sb, pa, pb, ra, rb. Asignamos costes a cada movimiento de la siguiente manera:

sa, sb: 1 (intercambiar dos elementos es fácil)
pa, pb: 2 (mover un elemento entre pilas es moderado)
ra, rb: 3 (rotar una pila es más costoso)

En el primer paso, la pila es [5, 2, 8, 3, 1, 4, 6]. Exploramos todos los posibles movimientos:

sa: intercambiar 5 y 2 -> [2, 5, 8, 3, 1, 4, 6] (coste: 1)
sb: no se puede aplicar (pila B está vacía)
pa: mover 5 a pila B -> [2, 8, 3, 1, 4, 6] y [5] (coste: 2)
pb: no se puede aplicar (pila B está vacía)
ra: rotar pila A -> [2, 8, 3, 1, 4, 5, 6] (coste: 3)
rb: no se puede aplicar (pila B está vacía)

Seleccionamos el movimiento con el menor coste, que es sa.

Actualizamos la pila a [2, 5, 8, 3, 1, 4, 6].

En el siguiente paso, exploramos de nuevo todos los posibles movimientos y seleccionamos el mejor. Y así sucesivamente hasta que se ordena la pila.

Ventajas

El algoritmo de costes es útil porque:

☺ Permite explorar todas las posibles soluciones y seleccionar la mejor.
☺ Es flexible y se puede adaptar a diferentes problemas de optimización combinatoria.
☺ Puede ser implementado de manera eficiente utilizando técnicas de programación dinámica.

LA TOMA DE DATOS

Estructura `s_stack`

La estructura `s_stack` es una representación de un nodo en una pila doblemente enlazada utilizada en el contexto de un algoritmo de ordenación. Esta estructura contiene varios campos que ayudan a gestionar y manipular los nodos de la pila durante el proceso de ordenación:

1. **`int value`**:
 - Almacena el valor del nodo en la pila.

2. **`int index`**:
   - Almacena el índice del nodo.

3. **`int cost`**:
   - Almacena el costo asociado con mover este nodo a su posición correcta en la pila ordenada. Este costo puede ser calculado en función de las operaciones necesarias para mover el nodo.

4. **`int pos`**:
   - Almacena la posición actual del nodo en la pila.

5. **`int target`**:
   - Almacena la posición objetivo del nodo en la pila ordenada.

6. **`int a_cost`**:
   - Almacena el costo de mover el nodo en la pila `a`.

7. **`int b_cost`**:
   - Almacena el costo de mover el nodo en la pila `b`.

8. **`struct s_stack *next`**:
   - Puntero al siguiente nodo en la pila. Esto permite la navegación hacia adelante en la pila.

9. **`struct s_stack *prev`**:
   - Puntero al nodo anterior en la pila. Esto permite la navegación hacia atrás en la pila.

10. **`struct s_stack *target_node`**:
    - Puntero al nodo objetivo en la pila ordenada. Esto puede ser utilizado para identificar el nodo que debería estar en la posición objetivo de este nodo.

11. **`bool above`**:
    - Un valor booleano que puede ser utilizado para indicar si el nodo está por encima de otro nodo en algún contexto específico del algoritmo.

12. **`bool best`**:
    - Un valor booleano que puede ser utilizado para marcar si este nodo es el mejor candidato para alguna operación específica en el algoritmo.

La estructura `s_stack` permite la implementación de una pila doblemente enlazada con información adicional que es útil para algoritmos de ordenación complejos. Aquí hay un desglose de cómo se utiliza cada campo en el contexto de un algoritmo de ordenación:

1. **Navegación en la Pila**:
   - Los punteros `next` y `prev` permiten recorrer la pila en ambas direcciones, lo que es útil para operaciones que requieren acceso a nodos adyacentes.

2. **Ordenación y Costos**:

- Los campos `cost`, `a_cost`, y `b_cost` permiten calcular y almacenar los costos asociados con mover nodos a sus posiciones correctas. Esto es crucial para algoritmos que optimizan el número de movimientos.

3. **Posicionamiento y Objetivos**:
- Los campos `pos` y `target` ayudan a rastrear la posición actual y la posición objetivo de cada nodo, lo que facilita la implementación de estrategias de ordenación.

4. **Marcadores y Estados**:
- Los campos booleanos `above` y `best` pueden ser utilizados para marcar nodos que cumplen ciertos criterios durante el proceso de ordenación, permitiendo decisiones más informadas.

En un algoritmo de ordenación como "push_swap", esta estructura se utiliza para:

1. **Inicializar la Pila**:
- Crear nodos con valores específicos y enlazarlos para formar la pila inicial.

2. **Calcular Costos**:
- Evaluar el costo de mover cada nodo a su posición correcta y almacenar estos costos en los campos correspondientes.

3. **Realizar Movimientos**:
- Utilizar los punteros `next` y `prev` para realizar operaciones de rotación y empuje en la pila.

4. **Optimizar el Ordenamiento**:
- Utilizar los campos `target_node`, `above`, y `best` para identificar y mover los nodos de manera óptima.

En resumen:

La estructura `s_stack` es una representación rica en información de un nodo en una pila doblemente enlazada, diseñada para soportar algoritmos de ordenación complejos. Contiene campos para valores, índices, costos, posiciones, punteros a nodos adyacentes y objetivos, así como marcadores booleanos para facilitar la implementación de estrategias de ordenación eficientes.

Recuerda: ¡El último en llegar, es el primero!

La función main

La función main hace lo siguiente:

- Verifica si se han proporcionado suficientes argumentos en la línea de comandos.
- Inicializa dos pilas vacías (stack_a y stack_b).
- Incluye los argumentos válidos en la pila stack_a utilizando la función ft_stacknew.
- Verifica si la pila stack_a tiene duplicados utilizando la función ft_be_duplicated.
- Verifica si la pila stack_a está ordenada utilizando la función be_sorted.
- Si la pila no está ordenada, llama a la función ft_sort para ordenarla.
- Libera la memoria utilizada por las pilas utilizando la función ft_free_all.

Cómo lo hace:

Supongamos que se llama al programa con los siguientes argumentos:

./push_swap 5 3 1 4 2.

1. La función be_nbr verifica si cada argumento es un número válido. Si no lo es, el programa termina con un error.
2. La función ft_stacknew crea un nuevo elemento de la pila con el valor del argumento y lo agrega al principio de la pila stack_a.
3. La pila stack_a queda así: 2 -> 4 -> 1 -> 3 -> 5.
4. La función ft_be_duplicated verifica si la pila tiene duplicados. Si encuentra un duplicado, el programa termina con un error.
5. La función be_sorted verifica si la pila está ordenada. Si no lo está, el programa llama a la función ft_sort para ordenarla.
6. La función ft_sort ordena la pila utilizando el algoritmo de ordenación correspondiente (se muestran más adelante).
7. La pila stack_a queda ordenada: 1 -> 2 -> 3 -> 4 -> 5.
8. Finalmente, el programa libera la memoria utilizada por las pilas utilizando la función ft_free_all.

push_swap.c

Este código es un programa que implementa un algoritmo de ordenación utilizando pilas. El programa verifica si los argumentos de la línea de comandos son números válidos, los incluye en una pila (stack_a), verifica si la pila tiene duplicados, y si la pila está ordenada. Si la pila no está ordenada, el programa llama a una función ft_sort para ordenarla.

El programa hace lo siguiente:

- Verifica si los argumentos de la línea de comandos son números válidos utilizando la función be_nbr.
- Incluye los argumentos válidos en una pila (stack_a) utilizando la función ft_stacknew.
- Verifica si la pila tiene duplicados utilizando la función ft_be_duplicated.
- Verifica si la pila está ordenada utilizando la función be_sorted.
- Si la pila no está ordenada, llama a la función ft_sort para ordenarla.
- Libera la memoria utilizada por las pilas utilizando la función ft_free_all.

Aquí hay un ejemplo de cómo funciona el programa:

Supongamos que se llama al programa con los siguientes argumentos: ./push_swap 5 3 1 4 2.

- La función be_nbr verifica si cada argumento es un número válido. Si no lo es, el programa termina con un error.
- La función ft_stacknew crea un nuevo elemento de la pila con el valor del argumento y lo agrega al principio de la pila stack_a.

- La función ft_be_duplicated verifica si la pila tiene duplicados. Si encuentra un duplicado, el programa termina con un error.
- La función be_sorted verifica si la pila está ordenada. Si no lo está, el programa llama a la función ft_sort para ordenarla.
- La función ft_sort ordena la pila utilizando un algoritmo de ordenación (lo veremos más adelante).
- Finalmente, el programa libera la memoria utilizada por las pilas, utilizando la función ft_free_all.

PASANDO VALORES A LA PILA B

La ventaja de pasar todos los valores a la pila B con el criterio de que en una primera vuelta (la mitad de la pila A) los valores deban cumplir tener un índice menor a la mitad del número total de elementos en la pila A es que se reduce significativamente el número de movimientos necesarios para ordenar la pila.

Al mover los elementos con índice menor a la mitad de la pila A a la pila B, se logra lo siguiente:

- ☺ Reducir la complejidad: Al mover los elementos con índice menor a la mitad de la pila A, se reduce la complejidad del problema, ya que se están eliminando los elementos que no están en su posición correcta en la pila A.

- ☺ Crear un orden parcial: Al mover los elementos con índice menor a la mitad de la pila A, se crea un orden parcial en la pila B, lo que facilita la ordenación posterior.

- ☺ Reducir el número de movimientos: Al mover los elementos con índice menor a la mitad de la pila A, se reduce el número de movimientos necesarios para ordenar la pila, ya que se están eliminando los elementos que no están en su posición correcta en la pila A.

En particular, si se considera que la pila A tiene n elementos, al mover los elementos con índice menor a la mitad de la pila A (es decir, n/2 elementos) a la pila B, se reduce el número de movimientos necesarios para ordenar la pila en un factor de n/2.

Por ejemplo, si se tiene una pila A con 10 elementos, al mover los 5 elementos con índice menor a la mitad de la pila A (es decir, los elementos con índice 0, 1, 2, 3 y 4) a la pila B, se reduce el número de movimientos necesarios para ordenar la pila en un factor de 5.

En resumen, la ventaja de pasar todos los valores a la pila B con el criterio de que en una primera vuelta (la mitad de la pila A) los valores deban cumplir tener un índice menor a la mitad del número total de elementos en la pila A es que se reduce significativamente el número de movimientos necesarios para ordenar la pila, lo que facilita la ordenación posterior.

ft_tsort_big.c

Este código continúa con la implementación de las funciones que se encargan de ordenar una pila de enteros utilizando los algoritmos de ordenamiento específicos utilizados en este programa.

La **función ft_pos_min** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- pos: un entero que representa la posición actual en la pila A

  La función devuelve la posición del elemento mínimo en la pila A.

Se verifica si la pila A es nula o vacía. Si es así, se devuelve -1.
Se inicializa una variable post en pos.
Se crea un puntero tmp que apunta a la cabeza de la pila A.
Se inicializa una variable min en el valor del primer elemento de la pila A.
Se itera sobre la pila A utilizando el puntero tmp. En cada iteración, se verifica si el valor del elemento actual es menor que min. Si es así, se actualiza min y post con el valor y la posición actuales, respectivamente.
Se devuelve post al final de la función.

Supongamos que la pila A es:
stack_a: 4 (índice 4) -> 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)
Llamamos a la función ft_pos_min con los parámetros stack_a y 0.
La función devuelve 2, que es la posición del elemento mínimo en la pila A.

La **función ft_sort_max,** a la que nos enviaba la función ft_tsort para entradas de argmentos con más de cinco valores, toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de mover los elementos más grandes de la pila A a la pila B.

- Se calcula la posición max como la mitad de la longitud de la pila A.
- Se itera sobre la pila A hasta que su longitud sea menor o igual a max + 1. En cada iteración, se verifica si el índice del elemento actual es mayor que max + 1. Si es así, se rota la pila A hacia la derecha. Si no, se mueve el elemento a la pila B.
- Se mueven todos los elementos restantes de la pila A a la pila B.
- Se llama a la función ft_sort_end para ordenar la pila A y la pila B.

La función **ft_sort_end** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de ordenar la pila A y la pila B utilizando diferentes algoritmos de ordenamiento dependiendo de la posición de los elementos en la pila A.

- Se llama a la función ft_sort_min para ordenar la pila A.
- Se itera sobre la pila B hasta que esté vacía. En cada iteración, se llama a la función ft_find_pos_target para encontrar la posición del elemento actual en la pila B en la pila A.

- Se llama a la función ft_get_cost para calcular el costo de mover el elemento actual de la pila B a su posición correcta en la pila A.
- Se llama a la función ft_cheap_sort para mover el elemento actual de la pila B a su posición correcta en la pila A de manera óptima.
- Si la pila A no está ordenada al final, se llama a la función shifte_change para reordenar la pila A.

La **función ft_moves** toma cuatro parámetros:

- s_a: un puntero a un puntero a una pila A
- s_b: un puntero a un puntero a una pila B
- a_cost: un entero que representa el costo de mover un elemento en la pila A
- b_cost: un entero que representa el costo de mover un elemento en la pila B

La función se encarga de mover los elementos en la pila A y la pila B de manera óptima.

- Se verifica si ambos costos son negativos. Si es así, se llama a la función rrr_rot para rotar ambas pilas hacia la derecha.
- Se verifica si ambos costos son positivos. Si es así, se llama a la función rr_rot para rotar ambas pilas hacia la izquierda.
- Se llama a la función a_rot para rotar la pila A hacia la izquierda o derecha dependiendo del costo a_cost.
- Se llama a la función b_rot para rotar la pila B hacia la izquierda o derecha dependiendo del costo b_cost.
- Se mueve el elemento de la pila B a la pila A utilizando la función pa.

Supongamos que la pila A es:
stack_a: 4 (índice 4) -> 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2)
Y la pila B es:
stack_b: 5 (índice 5)
Llamamos a la función ft_moves con los parámetros stack_a, stack_b, 2 y 1.
La función mueve el elemento 5 de la pila B a su posición correcta en la pila A, que es entre los elementos 4 y 3.
La pila A queda ordenada:
stack_a: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4) -> 5 (índice 5)

ft_tsort_c.c

La **función ft_sort_min** toma un parámetro:

- stack: un puntero a un puntero a una pila

La función se encarga de ordenar la pila de manera que el elemento con el índice más alto esté en la parte superior de la pila.

- Se verifica si la pila ya está ordenada utilizando la función be_sorted. Si es así, se devuelve inmediatamente.

- Se encuentra el índice más alto en la pila utilizando la función ft_hi_index.
- Se verifica si el elemento con el índice más alto está en la parte superior de la pila. Si es así, se rota la pila hacia la derecha utilizando la función ra.
- Se verifica si el elemento con el índice más alto está en la segunda posición de la pila. Si es así, se rota la pila hacia la izquierda utilizando la función rra.
- Se verifica si el elemento en la parte superior de la pila es mayor que el elemento en la segunda posición. Si es así, se intercambian los dos elementos utilizando la función sa.

  Supongamos que la pila es:
  stack: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)
  Llamamos a la función ft_sort_min con el parámetro stack.
  La función ordena la pila de manera que el elemento con el índice más alto esté en la parte superior de la pila.
  La pila queda ordenada:
  stack: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4)

La **función shifte_change** toma un parámetro:

- stack_a: un puntero a un puntero a una pila A

  La función se encarga de reordenar la pila A de manera que el elemento con el índice más bajo esté en la parte superior de la pila.

- Se encuentra la posición del elemento con el índice más bajo en la pila utilizando la función ft_low_index.
- Se verifica si la posición del elemento con el índice más bajo es mayor que la mitad de la longitud de la pila. Si es así, se rota la pila hacia la izquierda utilizando la función rra hasta que el elemento esté en la parte superior de la pila.
- Si no, se rota la pila hacia la derecha utilizando la función ra hasta que el elemento esté en la parte superior de la pila.

  Supongamos que la pila A es:
  stack_a: 4 (índice 4) -> 3 (índice 3) -> 2 (índice 2) -> 1 (índice 1)
  Llamamos a la función shifte_change con el parámetro stack_a.
  La función reordena la pila A de manera que el elemento con el índice más bajo esté en la parte superior de la pila.
  La pila A queda ordenada:
  stack_a: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4)

La **función ft_low_index** toma un parámetro:

- stack: un puntero a un puntero a una pila

  La función se encarga de encontrar la posición del elemento con el índice más bajo en la pila.

- Se crea un puntero tmp que apunta a la cabeza de la pila.
- Se inicializa una variable low_index en el valor más alto posible (INT_MAX).

- Se itera sobre la pila utilizando el puntero tmp. En cada iteración, se verifica si el índice del elemento actual es menor que low_index. Si es así, se actualiza low_index y low_pos con el valor y la posición actuales, respectivamente.
- Se devuelve low_pos al final de la función.

  Supongamos que la pila es:
  stack: 4 (índice 4) -> 3 (índice 3) -> 2 (índice 2) -> 1 (índice 1)
  Llamamos a la función ft_low_index con el parámetro stack.
  La función devuelve la posición del elemento con el índice más bajo, que es 1.

La **función ft_hi_index** toma un parámetro:

- stack: un puntero a una pila

  La función se encarga de encontrar el índice más alto en la pila.

- Se crea una variable index que se inicializa con el índice del elemento en la cabeza de la pila.
- Se itera sobre la pila utilizando un puntero stack. En cada iteración, se verifica si el índice del elemento actual es mayor que index. Si es así, se actualiza index con el valor actual.
- Se devuelve index al final de la función.

  Supongamos que la pila es:
  stack: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3) -> 4 (índice 4)
  Llamamos a la función ft_hi_index con el parámetro stack.
  La función devuelve el índice más alto, que es 4.

En resumen, este código define varias funciones que se encargan de ordenar pilas de enteros utilizando algoritmos de ordenamiento específicos.
Las funciones ft_sort_min, shifte_change, ft_low_index y ft_hi_index se encargan de ordenar la pila de manera que el elemento con el índice más alto esté en la parte superior de la pila y el elemento con el índice más bajo esté en la parte inferior de la pila.

ft_tpos.c

Este código define varias funciones que se encargan de asignar posiciones y objetivos a elementos en dos pilas, stack_a y stack_b.

La función ft_add_pos toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de asignar una posición a cada elemento en ambas pilas.

- Se crea un puntero tmp_a que apunta a la cabeza de la pila A.
- Se crea un puntero tmp_b que apunta a la cabeza de la pila B.
- Se inicializan dos variables pos_a y pos_b en 0.

- Se itera sobre la pila A utilizando el puntero tmp_a. En cada iteración, se asigna la posición actual (pos_a) al elemento actual y se incrementa pos_a.
- Se itera sobre la pila B utilizando el puntero tmp_b. En cada iteración, se asigna la posición actual (pos_b) al elemento actual y se incrementa pos_b.
- Se llama a la función ft_free_all para liberar la memoria utilizada por los punteros tmp_a y tmp_b.

  Supongamos que la pila A es:
  stack_a: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3)
  Y la pila B es:
  stack_b: 4 (índice 4) -> 5 (índice 5) -> 6 (índice 6)
  Llamamos a la función ft_add_pos con los parámetros stack_a y stack_b.
  La función asigna posiciones a cada elemento en ambas pilas:
  stack_a: 1 (índice 1, posición 0) -> 2 (índice 2, posición 1) -> 3 (índice 3, posición 2)
  stack_b: 4 (índice 4, posición 0) -> 5 (índice 5, posición 1) -> 6 (índice 6, posición 2)

La **función ft_find_pos_target** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de encontrar la posición objetivo para cada elemento en la pila B en la pila A.

- Se llama a la función ft_pos_target para asignar posiciones a cada elemento en ambas pilas.
- Se crea un puntero tmp_b que apunta a la cabeza de la pila B.
- Se itera sobre la pila B utilizando el puntero tmp_b. En cada iteración, se llama a la función ft_target para encontrar la posición objetivo para el elemento actual en la pila A.
- Se asigna la posición objetivo encontrada al elemento actual en la pila B.
- Se llama a la función ft_free para liberar la memoria utilizada por el puntero tmp_b.

La **función ft_target** toma cuatro parámetros:

- stack_a: un puntero a un puntero a una pila A
- index: el índice del elemento en la pila B para el que se busca la posición objetivo
- target: el valor actual de la posición objetivo
- pos: la posición actual en la pila A

  La función se encarga de encontrar la posición objetivo para el elemento en la pila B en la pila A.

- Se crea un puntero tmp_a que apunta a la cabeza de la pila A.
- Se itera sobre la pila A utilizando el puntero tmp_a. En cada iteración, se verifica si el índice del elemento actual es mayor que el índice del elemento en la pila B y menor que el valor actual de la posición objetivo. Si es así, se actualiza el valor de la posición objetivo y la posición actual.
- Si no se encontró una posición objetivo en el paso anterior, se itera sobre la pila A nuevamente para encontrar la posición objetivo más cercana.
- Se devuelve la posición objetivo encontrada.

Supongamos que la pila A es:
stack_a: 1 (índice 1, posición 0) -> 2 (índice 2, posición 1) -> 3 (índice 3, posición 2) -> 4 (índice 4, posición 3)
Y la pila B es:
stack_b: 5 (índice 5, posición 0) -> 3 (índice 3, posición 1) -> 1 (índice 1, posición 2)
Llamamos a la función ft_find_pos_target con los parámetros stack_a y stack_b.
La función asigna posiciones objetivo a cada elemento en la pila B:
stack_b: 5 (índice 5, posición objetivo 3) -> 3 (índice 3, posición objetivo 2) -> 1 (índice 1, posición objetivo 0)

La **función ft_pos_target** toma un parámetro:

- stack: un puntero a un puntero a una pila

  La función se encarga de asignar posiciones a cada elemento en la pila.

- Se crea un puntero tmp que apunta a la cabeza de la pila.
- Se itera sobre la pila utilizando el puntero tmp. En cada iteración, se asigna la posición actual (pos) al elemento actual y se incrementa pos.

  Supongamos que la pila es:
  stack: 1 (índice 1) -> 2 (índice 2) -> 3 (índice 3)
  Llamamos a la función ft_pos_target con el parámetro stack.
  La función asigna posiciones a cada elemento en la pila:
  stack: 1 (índice 1, posición 0) -> 2 (índice 2, posición 1) -> 3 (índice 3, posición 2)

En resumen, este código define varias funciones que se encargan de asignar posiciones y objetivos a elementos en dos pilas. Las funciones ft_add_pos, ft_find_pos_target, ft_target y ft_pos_target se encargan de asignar posiciones y objetivos a los elementos en las pilas de manera que se pueda encontrar la posición objetivo para cada elemento en la pila B en la pila A.

ft_tmoves.c

Este código define dos funciones: ft_move_a y ft_select_a. Estas funciones se encargan de mover elementos en una pila A y una pila B de manera que se pueda ordenar la pila A de manera eficiente.

La **función ft_move_a** toma cuatro parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B
- pos: la posición objetivo en la pila A
- i: un indicador que determina la dirección del movimiento (0 para mover hacia arriba, 1 para mover hacia abajo)

  La función se encarga de mover el elemento en la posición pos en la pila A hacia arriba o hacia abajo dependiendo del valor de i.

- Si i es 1, se itera sobre la pila A utilizando un bucle while que se ejecuta hasta que la posición pos sea alcanzada. En cada iteración, se verifica si la pila B no está vacía y si el índice del elemento en la pila B es menor que el índice del último elemento en la pila B y el índice del elemento en la pila A es mayor que el índice del último elemento en la pila A. Si es así, se llama a la función rrr para rotar ambas pilas hacia abajo. De lo contrario, se llama a la función rra para rotar la pila A hacia abajo.
- Si i es 0, se itera sobre la pila A utilizando un bucle while que se ejecuta hasta que la posición pos sea alcanzada. En cada iteración, se llama a la función ra para rotar la pila A hacia arriba.

  Supongamos que la pila A es:
  stack_a: 3 (índice 3) -> 2 (índice 2) -> 1 (índice 1) -> 4 (índice 4)
  Y la pila B es:
  stack_b: vacía
  Llamamos a la función ft_move_a con los parámetros stack_a, stack_b, 2 y 1.
  La función mueve el elemento en la posición 2 en la pila A hacia abajo:
  stack_a: 3 (índice 3) -> 1 (índice 1) -> 2 (índice 2) -> 4 (índice 4)

La **función ft_select_a** toma dos parámetros:

- stack_a: un puntero a un puntero a una pila A
- stack_b: un puntero a un puntero a una pila B

  La función se encarga de seleccionar el elemento más cercano al centro de la pila A y moverlo hacia arriba o hacia abajo dependiendo de su posición.

- Se inicializan variables pos_a y pos_b en 0.
- Se calcula la longitud de la pila A y se divide entre 110 para obtener un valor de len.
- Se itera sobre la pila A utilizando un bucle while que se ejecuta hasta que se encuentre el elemento más cercano al centro de la pila A. En cada iteración, se llama a la función ft_find_pos_min para encontrar la posición del elemento más cercano al centro de la pila A.
- Se itera sobre la pila A nuevamente utilizando un bucle while que se ejecuta hasta que se encuentre el elemento más cercano al final de la pila A. En cada iteración, se llama a la función ft_find_pos_min para encontrar la posición del elemento más cercano al final de la pila A.
- Se compara la posición pos_a con la posición pos_b y se llama a la función ft_move_a con los parámetros stack_a, stack_b, pos_a y 0 si pos_a es menor que ft_stack_len(*stack_a) - pos_b, o con los parámetros stack_a, stack_b, pos_b y 1 de lo contrario.

  Supongamos que la pila A es:
  stack_a: 3 (índice 3) -> 2 (índice 2) -> 1 (índice 1) -> 4 (índice 4)
  Y la pila B es:
  stack_b: vacía
  Llamamos a la función ft_select_a con los parámetros stack_a y stack_b.
  La función calcula la longitud de la pila A y divide entre 110 para obtener un valor de len igual a 1.
  La función itera sobre la pila A utilizando un bucle while que se ejecuta hasta que se encuentre el elemento más cercano al centro de la pila A. En cada iteración, se llama a la función ft_find_pos_min para encontrar la posición del elemento más cercano al centro de la pila A.
  Supongamos que la función ft_find_pos_min devuelve la posición 2.

La función itera sobre la pila A nuevamente utilizando un bucle while que se ejecuta hasta que se encuentre el elemento más cercano al final de la pila A. En cada iteración, se llama a la función ft_find_pos_min para encontrar la posición del elemento más cercano al final de la pila A.
Supongamos que la función ft_find_pos_min devuelve la posición 3.
La función compara la posición pos_a (2) con la posición pos_b (3) y llama a la función ft_move_a con los parámetros stack_a, stack_b, 2 y 0.
La función ft_move_a mueve el elemento en la posición 2 en la pila A hacia arriba:
stack_a: 2 (índice 2) -> 3 (índice 3) -> 1 (índice 1) -> 4 (índice 4)

En resumen, este código define dos funciones que se encargan de mover elementos en una pila A y una pila B de manera que se pueda ordenar la pila A de manera eficiente.

La función ft_move_a mueve un elemento en la pila A hacia arriba o hacia abajo dependiendo de su posición y la función ft_select_a selecciona el elemento más cercano al centro de la pila A y lo mueve hacia arriba o hacia abajo dependiendo de su posición.

ft_tools.c

Este código define cinco funciones auxiliares que se encargan de manejar pilas y realizar operaciones básicas con ellas.

La **función ft_new_stack** toma un parámetro value y devuelve un puntero a un nuevo elemento de pila (t_stack *) con el valor especificado.

- Se reserva memoria para un nuevo elemento de pila utilizando malloc.
- Se verifica si la memoria se reservó correctamente. Si no es así, se llama a la función ft_error para mostrar un mensaje de error y salir del programa.
- Se asigna el valor value al campo value del nuevo elemento de pila.
- Se asigna NULL al campo next del nuevo elemento de pila, lo que indica que no hay un elemento siguiente en la pila.
- Se devuelve el puntero al nuevo elemento de pila.

Supongamos que llamamos a la función ft_new_stack con el parámetro 5. La función devuelve un puntero a un nuevo elemento de pila con el valor 5 y next igual a NULL.
t_stack *new_stack = ft_new_stack(5);

La **función ft_count_neg** toma un parámetro stack_a y devuelve el número de elementos en la pila stack_a que tienen un valor negativo.

- Se inicializa una variable count en 0.
- Se itera sobre la pila stack_a utilizando un bucle while que se ejecuta hasta que se llegue al final de la pila.
- En cada iteración, se verifica si el valor del elemento actual es negativo. Si es así, se incrementa la variable count.
- Se devuelve la variable count.

Supongamos que la pila stack_a es:

stack_a: 3 -> -2 -> 1 -> -4 -> 5
Llamamos a la función ft_count_neg con la pila stack_a. La función devuelve el valor 2, que es el número de elementos negativos en la pila.

La **función ft_b_to_a** toma dos parámetros stack_a y stack_b y mueve todos los elementos de la pila stack_b a la pila stack_a.

- Se itera sobre la pila stack_b utilizando un bucle while que se ejecuta hasta que la pila stack_b esté vacía.
- En cada iteración, se llama a la función pa para mover el elemento superior de la pila stack_b a la pila stack_a.
- Una vez que la pila stack_b esté vacía, se llama a la función ft_free para liberar la memoria reservada para la pila stack_b.

Supongamos que la pila stack_a es:
stack_a: 3 -> 1 -> 5
Y la pila stack_b es:
stack_b: 2 -> -4
Llamamos a la función ft_b_to_a con las pilas stack_a y stack_b. La función mueve todos los elementos de la pila stack_b a la pila stack_a, resultando en:
stack_a: 3 -> 1 -> 5 -> 2 -> -4

La función ft_abs toma un parámetro a y devuelve su valor absoluto.

- Se verifica si el valor a es negativo. Si es así, se devuelve el valor absoluto de a multiplicando por -1.
- De lo contrario, se devuelve el valor a sin cambios.

Supongamos que llamamos a la función ft_abs con el parámetro -5. La función devuelve el valor 5.
Función ft_error
La función ft_error toma un parámetro i y muestra un mensaje de error y sale del programa si i es menor que 1.

Se verifica si i es menor que 1. Si es así, se muestra un mensaje de error utilizando ft_printf.
Se sale del programa utilizando exit.

CHECKER (PARTE BONUS)

El objetivo es desarrollar un programa llamado checker, que tome como argumento el stack a en forma de una lista de enteros.

El primer argumento debe estar encima del stack (cuidado con el orden). Si no se da argumento, checker termina y no muestra nada.

Cuando todas las instrucciones se hayan leído, checker las ejecutará utilizando el stack recibido como argumento. Si tras ejecutar todas las instrucciones, el stack a está ordenado y el stack b vacío, entonces el programa checker mostrará “OK” seguido de un ’\n’ en el stdout.

En cualquier otro caso, deberá mostrar “KO” seguido de un ’\n’ en la salida estándar.

En caso de error, deberás mostrar Error seguido de un ’\n’ en la salida estándar.

Los errores incluyen, por ejemplo, que algunos o todos los argumentos no son enteros, que algunos o todos los argumentos sean más grandes que un número entero, que haya duplicados, o que una instrucción no exista y/o no tenga el formato correcto.

El código que acompaña a este documento en el correspondiente repositorio de github, se explica de la siguiente forma:

Función main

La función main es la entrada al programa.

Hace lo siguiente:

- Inicializa dos pilas vacías (stack_a y stack_b).
- Lee los argumentos de la línea de comandos y los incluye en la pila stack_a utilizando la función ft_include.
- Lee operaciones desde la entrada estándar (STDIN) utilizando la función get_next_line.
- Aplica cada operación a la pila stack_a utilizando la función ft_operations.
- Verifica si la pila stack_a está ordenada y vacía la pila stack_b utilizando la función be_sorted.
- Imprime "OK" si la pila stack_a está ordenada y vacía la pila stack_b, o "KO" en caso contrario.
- Libera la memoria utilizada por las pilas utilizando la función ft_free_all.

Función ft_include

Esta función inicializa la pila A con los valores pasados como argumentos al programa. Recorre los argumentos y, para cada uno, crea un nuevo elemento en la pila A con el valor correspondiente. Luego, verifica si la pila A está ordenada utilizando la función 'be_sorted'. Si no está ordenada, llama a la función 'ft_operations' para realizar las operaciones necesarias para ordenar la pila.

Cómo lo hace:

La función utiliza dos variables: 'tmp' e 'i'. 'tmp' es un puntero a un elemento de la pila, e 'i' es un contador que itera sobre los argumentos de la línea de comandos.

Supongamos que la función se llama con los siguientes argumentos: ./checker 5 3 1 4 2. La función iterará sobre los argumentos de la siguiente manera:

1. 'i' es igual a argc - 1, que es 5.
2. La función llama a 'be_nbr' para verificar si el argumento argv[i] es un número. Si no lo es, la función no hace nada más.
3. La función crea un nuevo elemento de la pila utilizando ft_checker_stacknew y ft_atoi para convertir el argumento en un entero.
4. La función asigna el nuevo elemento a tmp y lo agrega al principio de la pila stack_a.
5. La función decrementa 'i', y repite los pasos 2-4 hasta que 'i' es 0.

Una vez que la función ha incluido todos los argumentos en la pila, llama a la función ft_be_duplicated para verificar si hay duplicados en la pila. Si hay duplicados, la función no hace nada más.
Luego, la función llama a be_sorted para verificar si la pila está ordenada. Si la pila no está ordenada, la función tampoco hará nada.

Función ft_operations

Esta función es la encargada de realizar las operaciones sobre las pilas. Recibe tres parámetros: stack_a y stack_b: punteros a las pilas A y B, respectivamente.

Por otra parte, 'operation' es una cadena que representa la operación a realizar.

La función utiliza una serie de condicionales para determinar qué operación realizar según el valor de 'operation'. Por ejemplo, si 'operation' es "sa", la función llama a la función 'sa' (que se explicará más adelante) con el objetivo de intercambiar los dos primeros elementos de la pila A.

Si la cadena 'operation' no coincide con ninguna de las operaciones permitidas, la función imprimirá "Error\n", utilizando la función 'ft_printf'.

Función ft_be_duplicated

Esta función verifica si hay elementos duplicados en la pila A. Recorre la pila A y, para cada elemento, verifica si hay un elemento con el mismo valor en el resto de la pila. Si encuentra un elemento duplicado, llama a la función 'ft_error' (que se explicará más adelante) para manejar el error.

Cómo lo hace:

La función utiliza dos punteros: stack_a y tmp. stack_a apunta al primer elemento de la pila, y tmp apunta al elemento siguiente a stack_a.

- Supongamos que la pila stack_a tiene los siguientes elementos: 1 -> 2 -> 3 -> 2 -> 4.
- La función itera sobre la pila de la siguiente manera:
  - ✓ stack_a apunta al elemento 1, y tmp apunta al elemento 2.
  - ✓ La función verifica si el valor de stack_a (1) es igual al valor de tmp (2). No lo es, así que continúa.
  - ✓ tmp avanza al siguiente elemento (3).
  - ✓ La función verifica si el valor de stack_a (1) es igual al valor de tmp (3). No lo es, así que continúa.
  - ✓ tmp avanza al siguiente elemento (2).
  - ✓ La función verifica si el valor de stack_a (1) es igual al valor de tmp (2). Sí lo es, así que llama a ft_error para manejar el error.

En resumen: la función va iterando sobre la pila hasta que encuentra un elemento duplicado o hasta que llega al final de la pila.

Función be_sorted

Esta función verifica si la pila A está ordenada en orden ascendente. Recorre la pila A y verifica si cada elemento es menor que el siguiente. Si encuentra un elemento que no cumple esta condición, devuelve 0. En el caso de que la pila está ordenada, nos devuelve 1.

Cómo lo hace:

Supongamos que la pila stack_a tiene los siguientes elementos: 1 -> 2 -> 3 -> 4 -> 5. La función iterará sobre la pila de la siguiente manera:

- tmp apunta al elemento 1.
- La función verifica si el valor de tmp (1) es menor que el valor del elemento siguiente (2). Sí lo es, así que avanza tmp al siguiente elemento (2).
- La función verifica si el valor de tmp (2) es menor que el valor del elemento siguiente (3). Sí lo es, así que avanza tmp al siguiente elemento (3).
- La función itera sobre la pila hasta que llega al final.

Función be_nbr

La función ‘be_nbr’ verifica si una cadena de caracteres (str) corresponde a un número entero válido. Si no lo es, llama a la función ‘ft_error’ para manejar el error.

Aquí hay un ejemplo de cómo funciona:

- ☺ Si la cadena es "-123", la función devuelve true porque es un número entero válido.
- ☹ Si la cadena es "123abc", la función llama a ‘ft_error’, dado que no es un número entero válido.

La función hace lo siguiente:

- Inicializa dos variables: ‘nbr’ (un número entero largo) y ‘neg’ (un entero que se utiliza para indicar si el número es negativo).
- Verifica si el primer carácter de la cadena es '-' o '+'. Si lo es, establece ‘neg’ en 1 si es '-' y avanza al siguiente carácter.
- Verifica si la cadena está vacía después de avanzar al siguiente carácter. Si lo está, llama a ‘ft_error’.
- Recorre la cadena y verifica si cada carácter es un dígito (0-9). Si no lo es, llama a ‘ft_error’.
- Si el carácter es '+', avanza al siguiente carácter.
- Convierte el carácter a un número entero y lo suma a ‘nbr’.
- Si ‘neg’ es 1, establece ‘nbr’ en su valor negativo.
- Verifica si ‘nbr’ está dentro del rango de un número entero (INT_MIN a INT_MAX). Si no lo está, llama a ‘ft_error’.

Función ft_checker_stacknew

La función ft_checker_stacknew crea un nuevo elemento en la pila con un valor determinado.

En resumen: si se llama a la función con el valor 123, crea un nuevo elemento en la pila con el valor 123.

En concreto, la función hace lo siguiente:

- Inicializa un puntero a un elemento de la pila (new) en NULL.
- Asigna memoria para un nuevo elemento de la pila utilizando malloc.
- Verifica si la asignación de memoria fue exitosa. Si no lo fue, llama a free para liberar la memoria y devuelve NULL.
- Establece el valor del nuevo elemento en el valor pasado como argumento.
- Establece el puntero al siguiente elemento en NULL.
- Devuelve el puntero al nuevo elemento.

EVALUACIÓN

Debemos tener muy presente que durante **no se permite ninguna falta de segmentación ni ninguna otra terminación inesperada, prematura, incontrolada o inesperada del programa**, de lo contrario la calificación final será 0. Por ello se debe utilizar la flag correspondiente (-Wall). Esta regla estará activa durante toda la defensa.

El código de **cumplir la norma**.

Pérdidas de memoria.

Se debe prestar atención a la cantidad de memoria utilizada por push_swap (usando el comando valgrind, por ejemplo) para detectar cualquier anomalía y asegurarse de que la memoria asignada se libere correctamente. Si hay una pérdida de memoria (o más), la calificación final es 0.

Podemos realizar las siguientes pruebas:

```
% valgrind ./push_swap 18 2 3 4 5 6 7 9

% valgrind --leak-check=full ./push_swap 18 2 3 4 5 6 7 9
```

La secuencia de números utilizados son sólo ejemplos. Estas pruebas deberían dar el mismo resultado positivo con cualquier otra.

Gestión de errores

En esta sección, evaluaremos la gestión de errores de push_swap. Si al menos uno falla, no se otorgarán puntos por esta sección. Pase a la siguiente.

- Ejecute push_swap con parámetros no numéricos. El programa debe mostrar "Error".

  Ejemplo: ./push_swap a

- Ejecute push_swap con un parámetro numérico duplicado. El programa debe mostrar "Error".

Ejemplo: ./push_swap 1 1

- Ejecute push_swap solo con parámetros numéricos, incluido uno mayor que MAXINT. El programa debe mostrar "Error".

Ejemplo: ./push_swap 3 2 1 2147483648

- Ejecute push_swap sin ningún parámetro. El programa no debe mostrar nada.

Ejemplo: ./push_swap

Push_swap - Prueba de identidad

Se evaluará el comportamiento de push_swap cuando se le proporciona una lista que ya ha sido ordenada. Obviamente, no deben darse errores.

- Ejecute el siguiente comando "$>./push_swap 42". El programa no debería mostrar nada.
- Ejecute el siguiente comando "$>./push_swap 2 3". El programa no debería mostrar nada.
- Ejecute el siguiente comando "$>./push_swap 0 1 2 3". El programa no debería mostrar nada.
- Ejecute el siguiente comando "$>./push_swap 0 1 2 3 4 5 6 7 8 9". El programa no debería mostrar nada.
- Ejecute el siguiente comando "$>./push_swap 'incluyendo aquí entre cero y nueve valores elegidos al azar, pero ordenados'. El programa no debería mostrar nada (instrucción 0).

Push_swap - Versión simple

Para estas pruebas, se utilizará el código binario del verificador que se encuentra en los archivos adjuntos al efecto.

- Ejecute "$>./push_swap 2 1 0 | wc -l". Se mostrará el tamaño de la lista de instrucciones, que deben ser dos o tres, a lo sumo.
- Ejecute “$>./push_swap 2 1 0 | ./checker_linux 2 1 0”. Debe mostrarse: "OK".
- Ejecute "$>./push_swap ‘Entre 0 y 3 valores elegidos aleatoriamente’ | wc -l". Se mostrará el tamaño de la lista de instrucciones, que deben ser entre cero y tres.
- Ejecute “$>./push_swap ‘Entre 0 y 3 valores elegidos aleatoriamente’ | ./checker_linux 'los mismos valores introducidos anteriormente’”. Debe mostrarse: "OK".

Otra versión sencilla

- Ejecute "$>./push_swap 1 5 2 4 3 | wc -l". Se mostrará el tamaño de la lista de instrucciones, que no deberá ser mayor a doce. Felicitaciones si el tamaño de la lista de instrucciones es ocho.
- Ejecute “$>./push_swap 1 5 2 4 3 | ./checker_linux 1 5 2 4 3”. Debe mostrarse: "OK".
- Ejecute "$>./push_swap ‘5 valores elegidos aleatoriamente’ | wc -l". Se mostrará el tamaño de la lista de instrucciones, que no debe ser superior a doce.
- Ejecute “$>./push_swap ‘5 valores elegidos aleatoriamente’ | ./checker_linux 'los mismos valores introducidos anteriormente’”. Debe mostrarse: "OK".

Push_swap - Versión intermedia

- Ejecute "$>./push_swap ‘100 valores elegidos aleatoriamente’ | wc -l". Se mostrará el tamaño de la lista de instrucciones.
- Ejecute “$>./push_swap ‘100 valores elegidos aleatoriamente’ | ./checker_linux 'los mismos valores introducidos anteriormente’”. Debe mostrarse: "OK".

- ☺ Menos de 700: 5 puntos.
- ☺ Menos de 900: 4 puntos.
- ☺ menos de 1100: 3 puntos.
- ☺ menos de 1300: 2 puntos.
- ☺ Menos de 1500: 1 punto.

Push_swap - Versión avanda

- Ejecute "$>./push_swap ‘500 valores elegidos aleatoriamente’ | wc -l". Se mostrará el tamaño de la lista de instrucciones.
- Ejecute “$>./push_swap ‘500 valores elegidos aleatoriamente’ | ./checker_linux 'los mismos valores introducidos anteriormente’”. Debe mostrarse: "OK".

- ☺ Menos de 5500: 5 puntos.
- ☺ menos de 7000: 4 puntos.
- ☺ menos de 8500: 3 puntos.
- ☺ menos de 10000: 2 puntos.
- ☺ Menos de 11500: 1 puntos.

Generadores de números aleatorios

https://pinetools.com/es/generador-numeros-aleatorios
https://push-swap-visualizer.vercel.app/

Parte Bonus

Programa de verificación: gestión de errores

Obviamente, el primer paso será compilar el checker. Y después:

- Ejecute el verificador con parámetros no numéricos. El programa debe mostrar "Error".
- Ejecute el verificador con un parámetro numérico duplicado. El programa debe mostrar "Error".
- Ejecute el verificador solo con parámetros numéricos, incluido uno mayor que MAXINT. El programa debe mostrar "Error".
- Ejecute el verificador sin ningún parámetro. El programa no debe mostrar nada.
- Ejecute el verificador con parámetros válidos y escriba una acción que no exista durante la fase de instrucción. El programa debe mostrar "Error".
- Ejecute el verificador con parámetros válidos y escriba una acción con uno o varios espacios antes y/o después de la acción durante la fase de instrucción. El programa debe mostrar "Error".

Programa de verificación: pruebas falsas

No olvides presionar CTRL+D para detener la lectura durante la fase de instrucción.

- Ejecute el verificador con el siguiente comando "$>./checker 0 9 1 8 2 7 3 6 4 5" y luego escriba la siguiente lista de acciones válidas "[sa, pb, rrr]". El verificador debería mostrar "KO".

- Ejecute el verificador con una lista válida como parámetro de su elección y luego escriba una lista de instrucciones válida, pero, que no ordene los números enteros. El verificador debería mostrar "KO".

Programa de verificación: Pruebas correctas

- No olvide presionar CTRL+D para detener la lectura durante la fase de instrucción.
- Ejecute el verificador con el siguiente comando "$>./checker 0 1 2" y luego presione CTRL+D sin escribir ninguna instrucción. El programa debería mostrar "OK".
- Ejecute el verificador con el siguiente comando "$>./checker 0 9 1 8 2" y luego escriba la siguiente lista de acciones válidas "[pb, ra, pb, ra, sa, ra, pa, pa]". El programa debería mostrar "OK".
- Ejecute el verificador con una lista válida como parámetro de su elección y luego escriba una lista de instrucciones válida que ordene los números enteros. El verificador debe mostrar "OK".