Push_swap — Apunte de Estudio Completo

Autor: ChatGPT (modo profesor) — Escuela 42 Madrid

Este documento te guiará desde los fundamentos de estructuras de datos en C hasta la comprensión e implementación completa del proyecto Push_swap. Está pensado para estudiar sin conexión, como un apunte de clase con teoría, ejemplos, y ejercicios prácticos. Todo el código se presenta en inglés, pero las explicaciones en español para favorecer el aprendizaje progresivo.

1. Data Structures Fundamentals

Un stack es una estructura LIFO (Last In, First Out). En C, podemos implementarlo fácilmente mediante listas enlazadas para evitar problemas de tamaño fijo. Una lista enlazada doble nos permite movernos hacia adelante y hacia atrás, lo que facilita operaciones como rotate y reverse rotate.

Estructuras básicas en C:

```
typedef struct s_node { int value; struct s_node *next; struct s_node *prev; }
t_node; typedef struct s_stack { t_node *top; t_node *bottom; int size; } t_stack;
```

Ejercicio 1: Crea un stack vacío y añade tres números, luego imprime su contenido.

2. Stack Manipulation

Push_swap utiliza dos stacks (a y b). Las operaciones básicas son: sa, sb, ss, pa, pb, ra, rb, rr, rra, rrb, rrr.

Ejemplo de push y pop:

```
void push(t_stack *stack, int value); int pop(t_stack *stack);
```

Ejercicio 2: Simula manualmente las operaciones pb, pa, ra, y rra sobre un stack con 5 números.

3. Sorting Algorithms

El objetivo del proyecto es ordenar el stack a utilizando el menor número posible de operaciones. Para pocos elementos, se puede usar lógica hardcodeada. Para listas grandes, estrategias como chunk sorting o radix sort.

Ejercicio 3: Diseña un algoritmo que ordene 5 números utilizando solo las operaciones básicas.

Ejercicio 4: Investiga el método Radix Sort y explica cómo podría aplicarse en Push_swap.

4. The push_swap Program

El programa debe recibir argumentos, validar que sean enteros únicos, y generar una secuencia óptima de instrucciones para ordenar el stack a.

Makefile flags: -Wall -Wextra -Werror Funciones permitidas: read, write, malloc, free, exit, ft_printf

Ejercicio 5: Escribe una función que valide si una cadena es un número entero válido.

Ejercicio 6: Implementa la función main que inicializa stack a con los argumentos recibidos.

5. Solutions & Tips

Consejos generales: - Usa listas dobles para mayor flexibilidad. - Mantén el código modular (funciones pequeñas y claras). - Verifica constantemente los leaks de memoria. - Testea cada operación individualmente antes de combinarlas. Solución Ejercicio 1: push y pop funcionan modificando punteros top/bottom. Solución Ejercicio 3: Para 3 o 5 números, prueba todas las combinaciones posibles y aplica la mínima secuencia. Solución Ejercicio 4: En radix sort, se usa el índice binario de cada número para decidir qué pasar a stack b en cada bit.