Roteiro de Estudo: Implementando o Built-in `env`

Visão Geral

O objetivo do `env` é simples: imprimir na saída padrão (stdout) todo o conteúdo da sua Hash Table de ambiente, com cada entrada formatada como `CHAVE=VALOR` e seguida por uma nova linha.

Para isso, precisamos de três componentes lógicos:
1. Um método para **iterar** sobre todas as entradas da sua Hash Table.
2. Um método para **formatar** cada par (chave, valor) na string `CHAVE=VALOR`.
3. Um método para **imprimir** essa string formatada na saída padrão.

Módulo 1: O Ponto de Partida - Como Iterar na sua Hash Table?

Objetivo: Desenvolver uma estratégia para "varrer" sua Hash Table e visitar cada par chave-valor armazenado, já que `env` precisa imprimir *todos* eles.

Exercício 1.1: O Desafio da Iteração

- **Foco:** O design interno da sua Hash Table.
- **Tarefa:** Seu `ht_search` é ótimo para encontrar *um* item, mas `env` precisa de *todos*. Você não pode usar `ht_search` para isso, pois você não sabe todas as chaves de antemão.

Provocação de Raciocínio:

1. Qual é a estrutura de dados interna da sua Hash Table? (Provavelmente um array de ponteiros, onde cada ponteiro é a cabeça de uma lista ligada para tratar colisões).
2. Como você faria um "dump" completo dessa estrutura? Você precisará de um loop `for` para varrer o array principal (os "buckets").
3. Dentro desse loop `for`, o que você fará se o ponteiro no bucket não for `NULL`? (Você precisará de um loop `while` aninhado para percorrer a lista ligada até o final).

Exercício 1.2: Esboçando o Iterador

- **Foco:** O algoritmo de "varredura total".
- **Tarefa:** Esboce a lógica (em pseudocódigo ou em C) para uma função `void ht_dump(t_hash_table *table)`. Dentro dessa função, implemente a lógica de loop duplo (o `for` pelos buckets e o `while` pelas listas ligadas) que, por enquanto, apenas imprime a chave de cada item que encontra.
- **Ponto de Consolidação:** Esta lógica de iteração é o motor do seu comando `env`.

Módulo 2: O Mecanismo de Saída - A Função `write`

Objetivo: Dominar a função `write` para imprimir dados na saída padrão.

Exercício 2.1: Imprimindo na Saída Padrão

- **Foco:** `write(int fd, const void *buf, size_t count)`
- **Tarefa:** Escreva um programa de teste simples que imprime "Hello, world!" na tela usando `write`.

Provocação de Raciocínio:

1. Qual é o `fd` (file descriptor) para a "saída padrão" (stdout)? (Dica: é `1`, ou a constante `STDOUT_FILENO` de `<unistd.h>`).
2. Para o argumento `buf`, você pode passar uma string literal `"Hello"`.
3. Para o argumento `count`, você não pode simplesmente passar `sizeof("Hello")`. Por quê? Qual função da sua `libft` você deve usar para obter o comprimento correto da string? (Dica: `ft_strlen`).

Exercício 2.2: O Desafio da Nova Linha

- **Foco:** `write` não adiciona automaticamente.
- **Tarefa:** Como você imprimiria `CHAVE=VALOR` e depois uma nova linha? Tente fazer isso no seu iterador do Módulo 1.

Abordagem Ineficiente (mas funcional):

1. `write(1, item->key, ft_strlen(item->key));`

2. `write(1, "=", 1);`
3. `write(1, item->value, ft_strlen(item->value));`
4. `write(1, " ", 1);`

- **Provocação:** Isso funciona, mas é eficiente? Você está fazendo quatro chamadas de sistema separadas para cada linha. Como você poderia reduzir isso para uma única chamada?

Módulo 3: A Solução Eficiente - Sua Dynamic String

Objetivo: Usar sua biblioteca de string dinâmica para construir a string de saída inteira em memória *antes* de imprimi-la, reduzindo 4 chamadas de sistema para apenas 1.

Exercício 3.1: Construindo a String de Saída

- **Foco:** As funções da sua biblioteca `dynamic_string.c`.
- **Tarefa:** Assumindo que sua biblioteca tenha funções como `str_new(char *)` (cria uma nova string dinâmica) e `str_append(char *dyn_str, char *suffix)` (anexa a uma string dinâmica):

Provocação de Raciocínio: Dentro do seu loop de iteração da Hash Table, como você construiria a string de saída `CHAVE=VALOR` em uma única variável `char *output_line` usando suas funções?

1. Como você inicializa a string com a chave?
2. Como você anexa o caractere `=`?
3. Como você anexa o valor?
4. Como você anexa a nova linha ?

Exercício 3.2: A Impressão Única

- **Foco:** Combinar Módulo 2 e Módulo 3.
- **Tarefa:** Agora que você tem a string completa em `output_line`, como você a imprime com uma única chamada `write`? (Lembre-se do Exercício 2.1: você precisa

do `fd`, do ponteiro `output_line`, e do `ft_strlen(output_line)`).

- **Provocação:** O que você deve fazer com a memória de `output_line` *depois* que `write` terminar de usá-la? (Dica: `str_free()` ou `free()`, dependendo da sua biblioteca).

Módulo 4: Integração Final e Casos de Borda

Objetivo: Montar a função `b_env` (built-in `env`) final.

Exercício 4.1: A Função `b_env`

- **Foco:** A estrutura final do built-in.
- **Tarefa:** Crie a função `int b_env(t_hash_table *env_table)` (ou qualquer que seja a assinatura do seu built-in). Implemente o loop de iteração (Módulo 1) e, dentro dele, a lógica de construção de string dinâmica (Módulo 3) e a impressão com `write` (Módulo 2).
- **Ponto de Consolidação:** Neste ponto, seu `env` deve funcionar perfeitamente quando chamado sozinho.

Exercício 4.2: Lidando com Argumentos (O Comportamento do `bash`)

- **Foco:** O que `env` deve fazer se receber argumentos?
- **Tarefa:** Teste o comando `env` no seu `bash` real. O que acontece se você digitar `env foo bar`?
- **Observação:** O `bash` (e o `env` real do sistema) tentará executar `foo bar` como um comando. **No entanto,** o `env` built-in do Minishell geralmente tem um requisito mais simples: apenas imprimir o ambiente.

Provocação de Raciocínio: Verifique os requisitos do seu projeto (`subject`).

1. Se `env` receber argumentos (ex: `cmd->args[1] != NULL`), qual deve ser o comportamento?
2. **Caminho A (Recomendado):** O `bash` built-in `env` (sem caminho, ex: `env`) na verdade não existe; ele usa o de `/usr/bin/env`. Mas seu `export` e `unset` *ignoram* argumentos extras silenciosamente. Talvez seu `env` deva fazer o mesmo: apenas

imprimir o ambiente, não importa quais argumentos sejam passados.

3. **Caminho B (Mais Correto, mas Simples):** Imprimir um erro, como "env: too many arguments" (ou nenhum erro, como `export`) e retornar um status de falha (`1`).

- **Decisão:** Para o `b_env`, a lógica mais simples e comum para o Minishell é ignorar argumentos extras e apenas executar a lógica de impressão. Verifique se isso é aceitável para o seu projeto.