Roteiro de Estudo: A Lógica da Expansão de Variáveis

Visão Geral

O objetivo é implementar a lógica de expansão que lida corretamente com todas as formas do caractere `$`. Assumimos que você recebe uma lista de tokens da fase de parsing. Nosso trabalho é iterar sobre esses tokens e, para cada um, analisar seu conteúdo de string e, se necessário, substituí-lo por um novo valor expandido. O foco será no algoritmo de análise e decisão.

Módulo 1: O Ponto de Partida - Análise do Token

Objetivo: Estabelecer o fluxo de controle principal e a lógica de decisão inicial para cada token.

Exercício 1.1: O Gatilho da Expansão

- **Foco:** Identificar quais tokens precisam de atenção.
- **Tarefa:** Projete a função controladora principal, `void expander(t_seu_tipo_de_comando *cmd)`. Ela deve iterar sobre cada token na lista de argumentos do comando.

Provocação de Raciocínio:

1. Para um token qualquer, qual é o caractere que sinaliza que uma expansão *pode* ser necessária?
2. Sua lógica deve procurar por este caractere. Se ele não estiver presente na string do token, você pode pular este token com segurança?
3. E as aspas simples? A expansão deve ocorrer em um token como `'Hello, $USER'`? Como a sua `struct` de token armazena a informação se ele estava ou não dentro de aspas simples? Sua lógica de iteração deve usar essa informação para decidir se deve ignorar completamente a expansão para um determinado token.

Módulo 2: Os Três Caminhos do `$`

Objetivo: Uma vez que você encontrou um `$` em um token que *pode* ser expandido, você precisa decidir o que ele significa. Existem três possibilidades distintas.

Exercício 2.1: A Bifurcação na Estrada

- **Foco:** A lógica de decisão baseada no caractere *imediatamente após* o `$`.
- **Tarefa:** Projete uma função ou um bloco de lógica que, ao encontrar um `$`, olha para o caractere seguinte para decidir qual caminho tomar.

Provocação de Raciocínio:

- Se o caractere seguinte for `?`, qual caminho você segue?
- Se o caractere seguinte for uma letra (a-z, A-Z) ou um underscore (`_`), qual caminho você segue?
- E se o caractere seguinte for qualquer outra coisa (um espaço, uma barra `/`, uma aspa, ou mesmo o final da string)? O que isso significa para o `$`? Ele é parte de uma variável ou apenas um caractere literal?

Exercício 2.2: Caminho A - A Expansão de `$?`

- **Foco:** Lidar com o status de saída.

Tarefa: Implemente a lógica para este caminho.

1. Obtenha o valor do status de saída (um `int`) do estado do seu shell.
2. Converta este `int` em uma nova string alocada.

- **Provocação:** Qual função da sua `libft` é ideal para a conversão de `int` para `string`? Quem se torna o dono da memória desta nova string e quem deve liberá-la?

Exercício 2.3: Caminho B - A Expansão de `$VAR`

- **Foco:** Lidar com variáveis de ambiente.

Tarefa: Implemente a lógica para este caminho.

1. A partir do caractere após o `$`, você precisa "consumir" todos os caracteres subsequentes que formam um nome de variável válido (letras, números, `_`). Projete um pequeno loop ou função que extrai este nome (ex: `"USER"`, `"PATH"`) para uma nova

string.

2. Use este nome extraído como chave para buscar na sua hash table.

3. Se a busca encontrar um valor, você o usará para a substituição.

• **Provocação:** E se a busca na hash table não encontrar a variável? Qual deve ser o valor de substituição para imitar o comportamento do `bash`?

Exercício 2.4: Caminho C - O `$` Literal

• **Foco:** Lidar com todos os outros casos.

• **Tarefa:** Se o caractere após o `$` não for `?` nem o início de um nome de variável válido, a expansão **não deve ocorrer**.

• **Provocação:** O que sua função de expansão deve fazer neste caso? A resposta é: nada. Ela deve continuar procurando pelo próximo `$` na mesma string do token. Isso significa que um token como `"Hello$ World"` permanecerá inalterado.

Módulo 3: A Cirurgia da String

Objetivo: Implementar a lógica de substituição que constrói uma nova string para o token.

Exercício 3.1: Reavaliando sua Ferramenta (`str_replace`)

• **Foco:** Entender a necessidade de uma substituição de *substring*.

• **Tarefa:** Considere um token cuja string seja `"PREFIX$VAR_SUFFIX"`. Seu parser pode gerar isso se não houver espaços.

Provocação de Raciocínio:

1. Você encontra `$VAR` e obtém seu valor, digamos `"VALUE"`.

2. Se você usar sua função `str_replace` atual, que apaga a string antiga e cria uma nova, o que acontece com `"PREFIX"` e `"_SUFFIX"`? Eles são perdidos.

3. Conclua que você precisa de um algoritmo mais poderoso que consiga construir uma nova string a partir de três pedaços: a parte **antes** da variável, o **valor** da

variável, e a parte **depois** da variável.

Exercício 3.2: Projetando a Substituição Robusta

- **Foco:** O algoritmo para construir a nova string.

Tarefa (Raciocínio): Descreva os passos para criar a nova string.

1. Como você calcula o tamanho total da memória que precisa alocar com `malloc`? Você precisará do comprimento das três partes.

2. Como você copiaria eficientemente as três partes para a nova área de memória na ordem correta? Pense nas funções da sua `libft` (`ft_memcpy`, `ft_strlcpy`, `ft_strlcat`).

- **Provocação:** E se um mesmo token tiver múltiplas variáveis, como `"$VAR1-$VAR2"`? Sua lógica de substituição deve ser colocada em um loop que continua procurando por mais `$` na string recém-construída até que não haja mais nenhum para expandir?

Módulo 4: A Integração Final

Objetivo: Aplicar sua nova lógica de expansão robusta a toda a sua AST e gerenciar a memória corretamente.

Exercício 4.1: Atualizando a Árvore

- **Foco:** Gerenciamento de memória e ponteiros.

- **Tarefa:** Sua função de expansão agora cria uma nova string alocada para o token.

Provocação de Raciocínio:

1. O que deve acontecer com a memória da string *antiga* do token? Ela deve ser liberada com `free()`.

2. Como o campo `val` (ou similar) da sua `struct` de token será atualizado para apontar para a nova string?

3. A sua função de expansão deve retornar a nova string ou modificar o ponteiro do token diretamente? Qual abordagem é mais limpa e segura?

Exercício 4.2: O Fluxo de Controle Final

- **Foco:** Juntar tudo.
- **Tarefa:** Trace o fluxo completo de uma entrada como `echo "Status: $?" '$HOME' $NO_VAR / $PWD`.

Ponto de Consolidação: Para cada token, descreva:

1. A decisão inicial do Módulo 1 (precisa de expansão?).
2. A decisão do caminho do Módulo 2 (é `$?`, `$VAR`, ou `$` literal?).
3. A operação de cirurgia de string do Módulo 3.
4. A atualização final do ponteiro do token do Módulo 4.

Se você conseguir traçar essa jornada para cada token, você terá dominado a lógica da expansão de variáveis.