Roteiro de Estudo: Expansão de Wildcard (`*`)

Visão Geral

O objetivo é ensinar o seu shell a interpretar o caractere `*` em um token como um padrão para corresponder nomes de arquivos no diretório atual (ou em um diretório especificado, como `src/*`).

Se atente para uma armadilha crucial: a expansão de `*` não é uma substituição de 1-para-1 (uma string por outra). É uma substituição de **1-para-N** (um token por *múltiplos* tokens).

Seu token original é `*.c`. Se houver 3 arquivos `.c` no diretório (`main.c`, `utils.c`, `parser.c`), seu token `*.c` deve ser *removido* e *substituído* por **três novos tokens** na lista de argumentos.

Caso nenhuma correspondência seja encontrada, o token original deve ser mantido.

Módulo 1: As Ferramentas Básicas - Lendo um Diretório

Objetivo: Aprender a listar o conteúdo de um diretório usando as funções permitidas. Por enquanto, vamos apenas *listar* tudo, sem nos preocupar com o `*`.

Exercício 1.1: Abrindo um Diretório (`opendir`)

- **Foco:** `opendir(const char *name)`
- **Tarefa:** Escreva um programa `list_files.c`. Na `main`, chame `opendir(".")`. O `.` é uma string que representa o diretório atual.

Provocação de Raciocínio:

1. O que a função `opendir` retorna? (Um ponteiro `DIR *`).
2. O que ela retorna se o diretório `.` não puder ser aberto (por exemplo, por falta de permissão)? (`NULL`).
3. Como você deve verificar se a chamada foi bem-sucedida antes de continuar?

Exercício 1.2: Lendo as Entradas (`readdir`)

- **Foco:** `struct dirent *readdir(DIR *dirp)`

- **Tarefa:** Modifique seu `list_files.c`. Após abrir o diretório com sucesso, crie um loop `while(1)`. Dentro do loop, chame `readdir` usando o ponteiro `DIR *` que você obteve.

Provocação de Raciocínio:

1. O que `readdir` retorna a cada chamada? (Um ponteiro para uma `struct dirent`).

2. O que ela retorna quando não há mais arquivos para ler no diretório? (`NULL`). Como você pode usar isso para quebrar seu loop `while` de forma segura?

3. A `struct dirent` tem um campo `char d_name[]`. Imprima este campo dentro do seu loop. O que você vê na tela?

Exercício 1.3: Fechando o Diretório (`closedir`)

- **Foco:** `int closedir(DIR *dirp)`

- **Tarefa:** Após o término do seu loop `while`, chame `closedir` com o seu ponteiro `DIR *`.

- **Provocação:** Por que esta função é importante? O que acontece com os recursos do seu programa (como "file descriptors") se você abrir muitos diretórios e nunca os fechar?

Exercício 1.4: Filtrando o Ruído

- **Foco:** O comportamento do `bash`.

- **Tarefa:** A saída do seu programa inclui `.` e `..` e, provavelmente, arquivos ocultos (como `.git`, `.DS_Store`). O `*` no `bash` **não** corresponde a arquivos que começam com um ponto (`.`).

- **Provocação:** Modifique seu loop `readdir` para que ele só imprima `d_name` se o primeiro caractere do nome do arquivo **não** for um ponto (`.`).

Módulo 2: O Algoritmo de Correspondência (O Mini-`fnmatch`)

Objetivo: Como `fnmatch` não é permitida, você precisa criar sua própria função que decide se uma string (`main.c`) corresponde a um padrão (`*.c`). Vamos construir uma versão simples que lida apenas com um `*`.

Exercício 2.1: O Desafio do Padrão

- **Foco:** O algoritmo de correspondência com `*`.
- **Tarefa:** Projete uma função `bool custom_match(const char *pattern, const char *text)`.

Provocação de Raciocínio:

1. Se o padrão for `*` (e apenas `*`), sua função deve sempre retornar `true` (lembrando que já filtramos os arquivos ocultos).
2. Se o padrão for `*.c` (um sufixo), como você verificaria se `text` termina com `.c`? (Dica: pense em `ft_strlen` e `ft_strncmp` ou `ft_strnrcmp` se você tiver).
3. Se o padrão for `src*` (um prefixo), como você verificaria se `text` começa com `src`? (Dica: `ft_strncmp`).
4. Se o padrão for `m*n` (prefixo e sufixo), como você verificaria se `text` começa com `m` E termina com `n`?
5. **Desafio Composto:** Tente escrever uma única função `custom_match` que lide com esses quatro casos básicos.

Módulo 3: A Expansão (O Problema 1-para-N)

Objetivo: Aplicar seu matcher e lidar com as duas consequências possíveis: 0 correspondências ou N correspondências.

Exercício 3.1: O Coletor de Correspondências

- **Foco:** Reunir os resultados.

Tarefa: Esboce a função principal `expand_wildcard(char *pattern)`. Esta função deve:

1. Abrir o diretório atual (`.`).

2. Ler cada entrada com `readdir`.
3. Filtrar entradas que começam com `.`.
4. Para cada entrada restante, chamar `custom_match(pattern, entry->d_name)`.
5. Se `custom_match` retornar `true`, armazenar uma **cópia** (usando `ft_strdup`) do `entry->d_name` em uma nova lista ligada ou array dinâmico.

• **Provocação:** Esta função deve retornar o quê? (Uma lista/array de strings correspondentes). E se nenhuma correspondência for encontrada, o que ela deve retornar? (Uma lista/array vazia ou `NULL`).

Exercício 3.2: O Dilema da Substituição

• **Foco:** O comportamento do shell: `ls *.c` vs `ls *.naoexiste`.

Tarefa: Agora, pense no seu `expander` principal, que itera sobre os tokens do comando.

1. Ele encontra um token que contém `*` (ex: `*.c`).
2. Ele chama sua função `expand_wildcard("*.c")`.

Provocação de Raciocínio (A Lógica Chave):

1. **Caso A:** A `expand_wildcard` retorna uma lista com 3 nomes (`main.c`, `utils.c`, `parser.c`). O que você deve fazer com o token `*.c` original na sua lista de argumentos? E como você insere os **três novos tokens** em seu lugar? (Isso é um desafio de manipulação de lista ligada).
2. **Caso B:** A `expand_wildcard` retorna uma lista vazia (`NULL`). De acordo com sua lógica, "ele apenas mantém o valor". O que você deve fazer com o token `*.c` original? (Exatamente: nada. Você o deixa em paz).

Módulo 4: O Toque Final - Lidando com Caminhos

Objetivo: Fazer sua lógica funcionar para padrões como `src/*.c`.

Exercício 4.1: Separando o Caminho do Padrão

- **Foco:** Análise da string de entrada.
- **Tarefa:** Seu token de entrada agora é `src/*.c`.

Provocação:

1. Qual diretório você deve abrir com `opendir`? (`.` ou `src`)?
2. Qual padrão você deve usar no seu `custom_match`? (`src/*.c` ou `*.c`)?

- **Conclusão:** Você precisa de uma lógica que separe o token `src/*.c` em duas partes: o **caminho do diretório** (`src`) e o **padrão de correspondência** (`*.c`). (Dica: `ft_strrchr` para encontrar a última `/`). Se não houver `/`, o caminho é `.` e o padrão é o token inteiro.

Exercício 4.2: Recompondo o Resultado

- **Foco:** A string de resultado final.
- **Tarefa:** Sua lógica do Exercício 4.1 abre `src` e usa o padrão `*.c` no `custom_match`. Ele encontra `main.c`.
- **Provocação:** Qual é a string que você deve adicionar à sua lista de tokens? É `main.c`? (Não). É `src/main.c`. Sua lógica precisa ser inteligente o suficiente para **reunir o caminho do diretório com o nome do arquivo correspondente** antes de criar o novo token.