Exercício Programa 3 - Estrutura de diretórios
MAC0122 Princípios de Desenvolvimento de Algoritmos - POLI

Prof. Ronaldo Fumio Hashimoto

1 Descrição do Exercício Programa

1.1 Visão Geral

Em sistemas operacionais UNIX, arquivos são armazenados usando a seguinte estrutura de diretórios (ou pastas): existe um diretório raiz, que pode conter arquivos e subdiretórios. Cada subdiretório, por sua vez, pode conter arquivos e subsubdiretórios. Cada subsubdiretório pode conter mais arquivos e subsubsubdiretórios, e assim sucessivamente. Por exemplo, a Fig. 1 abaixo mostra a estrutura de diretórios de uma pasta fictícia `ROOT`:

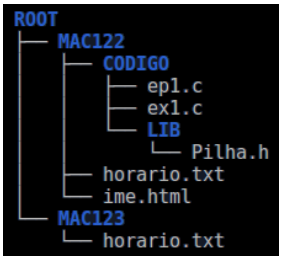

Figura 1: Exemplo da estrutura de diretórios da pasta ROOT, usando o comando tree do Linux. Subdiretórios estão escritos com cor azul, enquanto que arquivos estão escritos de cor branca.

Na Figura 1, a indentação indica a hierarquia dos diretórios. Por exemplo, a pasta `CODIGO` é subdiretório de `MAC122`, contém os arquivos `ep1.c` e `ex1.c` e também o diretório `LIB`.

O objetivo deste EP é simular o funcionamento de um sistema que permite a criação e a deleção de arquivos e diretórios.

1.2 Termos usados

A hierarquia de diretórios pode ser representada por uma **árvore**. O termo **elemento** será usado quando não for necessário fazer distinção entre um diretório ou um arquivo. Cada elemento da árvore será representado por um **nó**.

O termo **nível** será usado para definir a hierarquia dos nós. No exemplo da Fig. 1, vale que:

- A pasta `ROOT` se encontra no nível 0;
- Os subdiretórios `MAC122` e `MAC123`, contidos em `ROOT`, estão no nível 1;
- Os subdiretórios e arquivos contidos nas pastas de nível 1 estão no nível 2 e assim sucessivamente (na Fig. 1, estão no nível 2: a pasta `CODIGO`, o arquivo `ime.html` e ambos os arquivos `horario.txt`).

O nó no nível 0 é o nó **raiz**. Nós de nível 1 conectados à raiz são nós **filhos** da raiz, enquanto que a raiz é o **pai** destes nós. Analogamente, nós do nível 2 conectados a um nó $\mathcal{N}$ do nível 1 são filhos de $\mathcal{N}$ (e $\mathcal{N}$ é pai destes nós), e assim sucessivamente para os níveis subsequentes da árvore. Se dois nós têm o mesmo pai, dizemos que eles são nós **irmãos**.

Em computação, é comum representar árvores com a raiz no topo da árvore. Por exemplo, a Fig. 2 mostra uma representação em árvore da estrutura de diretórios da Fig. 1.

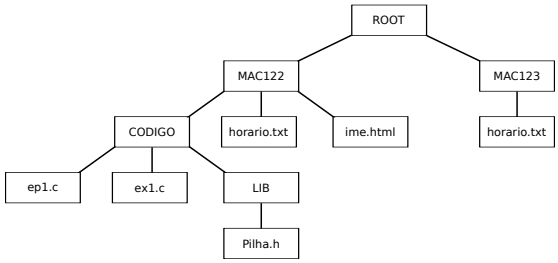

Figura 2: Representação da estrutura de diretórios da Fig. 1 em forma de árvore.

Veja que uma árvore é uma estrutura fundamentalmente recursiva: se removêssemos a raiz, teríamos duas subárvores de raízes `MAC122` e `MAC123`. Analogamente, se removêssemos os nós dos níveis 0 e 1, então os nós do nível 2 seriam raízes das subárvores restantes e assim sucessivamente.

2 Implementação

2.1 Representação

Naturalmente, cada nó será representado no computador usando uma `struct`. Em particular, diretórios e arquivos conterão os seguintes campos:

1. O nome do elemento (assumimos que o nome tem menos de 100 letras);
2. Um valor representando o tamanho gasto em memória daquele elemento. Vamos assumir que um diretório vazio tem tamanho 0 (zero).

Além destes campos, para representar as ligações entre os nós, ponteiros serão usados.

Como o número de filhos varia de acordo com o nó, usar um número fixo de ponteiros não é uma boa ideia. Assim, uma forma de se implementar as ligações com os filhos consiste da seguinte estratégia:

1. A partir do nó pai, crie um ponteiro para o primeiro filho (seta azul na Fig. 3).
2. A partir deste primeiro filho, use uma lista ligada para ligar todos os nós irmãos (região pontilhada na Fig. 3).

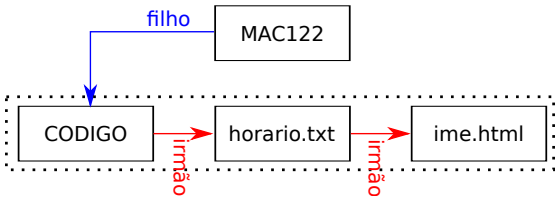

Figura 3: Representação do conjunto de nós MAC122 e seus filhos em memória. A seta azul indica o ponteiro para o primeiro filho. As setas vermelhas indicam o ponteiro para o próximo irmão. Ponteiros nulos não estão desenhados.

Assim sendo, cada nó também deverá ter dois ponteiros:

1. Um ponteiro para o primeiro filho;
2. Um ponteiro para o seu próximo irmão.

Para simplificar a implementação, sugere-se que seja usada um única `struct` para representar tanto um diretório quanto um arquivo. Neste sentido, pode-se adicionar um campo adicional à estrutura para indicar se o nó é um arquivo ou diretório.

2.2 Funções a serem implementadas

Para este EP, **um código cliente será dado**. Você deverá editar as linhas comentadas, implementando as seguintes operações requeridas pelo cliente:

1. **Criar um diretório:** Recebe como parâmetro o nome do diretório e cria um diretório vazio dentro do diretório atual.
2. **Criar um arquivo:** Recebe o nome do arquivo e o tamanho como parâmetros. Cria um arquivo com este nome e tamanho dentro do diretório atual.
3. **Mudar diretório atual:** Primeiramente, imprime os elementos no diretório atual (operação 5) para auxiliar o usuário. Então, pede para que o usuário digite o índice do diretório filho para o qual ele quer trocar.

   Por exemplo, na impressão do item 5 abaixo, digitar 0 trocaria o diretório atual para a pasta `CODIGO`.

4. **Voltar para o diretório raiz:** Troca o diretório atual para o diretório raiz.

5. **Imprimir os elementos no diretório atual:** Imprime os elementos filhos do diretório atual, precedidos do índice em que eles foram adicionados. O índice do primeiro elemento será 0, o índice do segundo elemento será 1 e assim sucessivamente. Para arquivos, o tamanho deles também deve ser impresso.

   Por exemplo, supondo que estamos na pasta `MAC122` do exemplo da Fig. 1, a saída esperada será:

   ```
   Arquivos em MAC122:
   (0)   CODIGO
   (1)   horario.txt (20 KB)
   (2)   ime.html (40 KB)
   ```

6. **Imprimir hierarquia do diretório atual:** Imprime todos os elementos na subárvore, indentados de acordo com o nível deles na hierarquia. Sua implementação deve usar recursão.

   Por exemplo, supondo que estamos na pasta `ROOT` do exemplo da Fig. 1, a saída esperada será:

   ```
   ROOT
     MAC122
       CODIGO
         ep1.c (60 KB)
         ex1.c (50 KB)
         LIB
           Pilha.h (10 KB)
       horario.txt (20 KB)
       ime.html (40 KB)
     MAC123
       horario.txt (30 KB)
   ```

   Foram usados 2 espaços para cada nível da árvore na impressão acima.

7. **Calcular o tamanho do diretório atual:** Calcula e imprime o tamanho acumulado de todos os elementos contidos na subárvore do diretório atual. Sua implementação deve usar recursão.

   Por exemplo, supondo que estamos na pasta `ROOT` no exemplo do item anterior, a saída esperada é:

   ```
   Tamanho do diretorio atual: 210 KB
   ```

   A saída esperada do diretório `CODIGO`, por sua vez, seria:

```
Tamanho do diretorio atual: 120 KB
```

8. **Apagar todos os elementos do diretório atual:** Deleta todos os arquivos e diretórios contidos na subárvore do diretório atual, mas não deleta o diretório atual. Sua implementação deve usar recursão.

   Por exemplo, se usarmos a operação 8 com o diretório `CODIGO` como o diretório atual, então o diretório `CODIGO` ficará vazio. Se, após deletarmos o diretório `CODIGO`, rodarmos as operações 4 e 6 (volta para o diretório raiz e imprime sua hieraraquia), a saída esperada será:

```
ROOT
  MAC122
    CODIGO
    horario.txt (20 KB)
    ime.html (40 KB)
  MAC123
    horario.txt (30 KB)
```

   O tamanho atualizado do diretório `ROOT`, usando a operação 7, também muda:

```
Tamanho do diretorio atual: 90 KB
```

3 Entrega do EP

3.1 Código

1. Você pode criar quantos arquivos achar necessário para implementar seu EP. Lembre-se, porém, de não violar o padrão interface/implementação/cliente.
2. Você pode mover os `define`s e funções do cliente para outros arquivos que você criar se achar conveniente.
3. Você pode modelar suas ADTs do jeito que achar melhor, porém essa modelagem será avaliada.
4. Sua solução deve utilizar algoritmo(s) recursivo(s) para calcular o nível e o tamanho das pastas.
5. O código deve estar claro e indentado, nomeie suas funções e variáveis com nomes que ajudem a entender seu código.
6. Libere-se de liberar a memória usada pelo seu programa antes de encerrar sua execução.

3.2 Pacote de Entrega

Você deve enviar todos os códigos-fonte (`.c` e `.h`) usados para a implementação do seu EP. Além disso, você deve enviar um arquivo `readme.txt` ou `readme.pdf` contendo as seguintes informações:

1. Descrição de suas ADTs
2. Sistema operacional utilizado para compilar o programa.
3. Programa usado para compilar os seus códigos-fonte (terminal / Codeblocks / XCode etc.)
4. Instruções de compilação e de uso do programa.

Todos os arquivos que não são código-fonte (`.c` ou `.h`) e projetos do Codeblocks devem estar descritos no seu arquivo de `readme.txt(.pdf)`.