Identificação:

Pedro Mesquita Maia – 2312664

Descrição:

O objetivo do trabalho foi desenvolver uma aplicação que gerencia uma lista encadeada de pacientes, onde cada nó da lista contém informações sobre o paciente, como ordem de chegada, status (E = Em espera, S = Saiu) e cor de prioridade (R = Vermelho, G = Verde, Y = Amarelo). O programa deve ser capaz de criar a lista de pacientes a partir de um arquivo de texto, imprimir a lista com contagem de cores, ordenar a lista pela ordem de chegada, adicionar novos pacientes, deletar nós específicos e também fazer a ordenação da lista.

Estrutura:

O código foi organizado em diversas funções além da `main`, cada uma com uma responsabilidade clara para facilitar a manipulação da lista de pacientes.

1. **createLinkedList**: Esta função é responsável por criar uma lista encadeada de pacientes lendo os dados de um arquivo de texto.
    - A função recebe um ponteiro `pHead` que representa a cabeça da lista.
    - Utiliza a função `fopen` para abrir o arquivo "db.txt" e, caso não seja possível abrir, imprime uma mensagem de erro e retorna `NULL`.
    - Em seguida, utiliza um loop para percorrer o arquivo e, para cada linha, lê os valores correspondentes ao status, ordem e cor do paciente. Um novo nó é criado para cada paciente e adicionado à lista.
    - Caso o arquivo tenha sido completamente lido, a função fecha o arquivo e retorna o ponteiro da lista encadeada.
    - Este processo garante que a lista esteja devidamente populada com os dados do arquivo.
2. **printLinkedList**: Esta função é responsável por imprimir a lista de pacientes na tela e também contar quantos pacientes existem de cada cor (R, G, Y).
    - A função percorre a lista desde a cabeça até o último nó, imprimindo a ordem, status e cor de cada paciente.
    - Durante a iteração, também faz uma contagem de quantos pacientes existem com cada cor (vermelho, verde e amarelo).
    - Ao final, a função exibe essa contagem de pacientes por cor.
    - Isso ajuda a visualizar o estado atual da lista e a distribuição de cores/prioridades dos pacientes.
3. **deleteNode**: Esta função é responsável por remover o nó correspondente ao paciente com uma determinada ordem.
    - A função recebe como parâmetros o ponteiro da cabeça da lista e a ordem do paciente que se deseja remover.

    - Se o paciente a ser removido for o primeiro nó da lista, a função simplesmente ajusta a cabeça da lista para o próximo nó.
    - Para nós intermediários ou finais, a função percorre a lista até encontrar o nó desejado, ajustando os ponteiros para que o nó seja removido corretamente da lista.
    - A função garante que, após a remoção, o nó de interesse seja liberado da memória.
    - Ao retornar o ponteiro atualizado da lista, o programa assegura que a lista mantenha a integridade após a remoção de qualquer nó.
4. **printIndividually**: Esta função imprime os detalhes de um único nó da lista, recebendo como parâmetro um ponteiro para o nó a ser impresso.
    - Exibe o status, a ordem e a cor do paciente armazenados no nó.
    - A função pode ser utilizada para depuração ou para exibir informações detalhadas de um único paciente.
5. **updateFile**: A função `updateFile` é responsável por salvar o estado atual da lista de pacientes de volta em um arquivo de texto, após operações de adição ou remoção de pacientes.
    - A função percorre toda a lista e escreve as informações de cada nó no arquivo "db.txt", substituindo o conteúdo anterior.
    - Isso garante que o estado da lista seja persistido no arquivo para uso futuro.
6. **addNode**: A função `addNode` adiciona um novo nó na lista encadeada.
    - Ela recebe como parâmetros a cabeça da lista e o nó que se deseja adicionar.
    - O novo nó é adicionado ao final da lista e, em seguida, a lista é reordenada pela ordem de chegada do paciente, utilizando a função `sortByOrder`.
    - Após a adição, a lista mantém a consistência e permanece ordenada.
7. **mergeSortedLists**: Esta função combina duas listas ordenadas em uma única lista ordenada.
    - A função utiliza a comparação entre as ordens dos pacientes de cada lista para determinar qual nó será adicionado primeiro.
    - Ela é chamada dentro da função `sortByOrder` para realizar a divisão e conquista na ordenação da lista.
8. **sortByOrder**: A função `sortByOrder` utiliza o método de divisão e conquista para ordenar a lista de pacientes com base na ordem de chegada.
    - Inicialmente, divide a lista em duas sublistas utilizando ponteiros de avanço rápido e lento.
    - Recursivamente, ordena cada sublista e depois combina as duas sublistas ordenadas utilizando a função `mergeSortedLists`.
9. **sortLinkedList**: Esta função é apenas um wrapper que chama a função `sortByOrder` para ordenar a lista completa pela ordem de chegada.

Solução:

Foram criadas duas constantes (`MIN = 0` e `MAX = 10000`) usadas para definir o tamanho das listas, laços de repetição e geração de números aleatórios (se necessário). A solução principal envolveu as seguintes etapas:

1. **Criação da Lista**: A função `createLinkedList` foi chamada para criar e popular a lista encadeada de pacientes a partir de um arquivo de texto.
2. **Ordenação Inicial**: Após a criação da lista, a função `sortLinkedList` foi utilizada para garantir que a lista estivesse ordenada pela ordem de chegada dos pacientes.
3. **Impressão da Lista**: A função `printLinkedList` foi utilizada para imprimir os pacientes da lista e contabilizar o número de pacientes por cor.
4. **Remoção de Nós**: A função `deleteNode` foi utilizada para remover nós específicos da lista com base na ordem de chegada. Após cada remoção, a lista foi novamente impressa.
5. **Adição de Novos Nós**: A função `addNode` foi usada para adicionar novos pacientes à lista, seguido de reordenação e impressão da lista atualizada.

Observações:

Houve algumas dificuldades iniciais na implementação da remoção de nós, especialmente ao lidar com a atualização de ponteiros, mas uma vez resolvido, o restante do código fluiu bem. A função de ordenação foi simplificada utilizando a estratégia de merge sort, o que garantiu uma ordenação eficiente.

O código foi compilado utilizando o comando `gcc -Wall -o main main.c` para verificar possíveis warnings, e o programa foi testado com sucesso em diferentes tamanhos de lista.