Projeto 1: Arquivos - Compactação e Recuperação dinâmica

O objetivo do projeto é gerenciar um sistema de arquivos que gerência a locação de veículos de uma dada empresa. O sistema armazena as seguintes informações:

- Código do Cliente (CodCli)
- Código do Veículo (CodVei)
- Nome do Cliente
- Nome do Veículo
- Número de Dias

A chave primária é composta pela composição "CodCli+CodVei". Para simplificar, considere que um cliente pode alugar uma única vez um determinado carro (i.e., CodCli+CodVei será sempre único). O arquivo a ser criado deve ser **binário** de **registros e campos de tamanho variável**, com **1 byte** no início do registro como **indicador de tamanho do registro**, e com campos separados por algum caractere especial (e.g., '#', '|', …).

Código do Cliente	Código do Veículo	Nome do Cliente	Nome do Veículo	Número de Dias
11 caracteres (fixo)	7 caracteres (fixo)	50 caracteres (máximo)	50 caracteres (máximo)	int (fixo ou caracteres)

Ex.: <57 *1 byte*>12121212121|ABC1234|João da Silva|Chevrolet Agile 2010|2|

As seguintes funcionalidades deverão estar disponíveis:

1. Inserção
2. Remoção
3. Compactação
4. Carrega Arquivos (dependente da implementação)

1 - Inserção

Ao adicionar um registro será preciso percorrer a lista de espaços disponíveis verificando se o novo registro se encaixa em algum dos espaços disponível (i.e., de algum registro removido). Para tanto, deve-se usar a estratégia ***first-fit*** e, para facilitar, podem considerar fragmentação interna (i.e., não precisa retornar para a Dispo o espaço que sobrou após a inserção de um registro no lugar de um que foi removido). Caso nenhum elemento da lista supra o espaço necessário para o novo registro, acrescente-o no final do arquivo.

Os dados a serem inseridos devem ser recuperados de um arquivo a ser fornecido no momento da execução do programa (vide funcionalidade 4).

2 - Remoção

Dada a chave de busca "CodCli+CodF" (recuperado de um arquivo a ser fornecido no momento da execução do programa – vide funcionalidade 4) realize a remoção do respectivo registro. A remoção deve ser feita diretamente no arquivo de dados.

Para reaproveitar o espaço removido, o arquivo deverá usar a seguinte estratégia para manter as informações dos arquivos que foram removidos:

(1) o arquivo deve ter um cabeçalho com um campo que indica o offset para o primeiro elemento da lista de espaços disponível, resultado da remoção de um registro.
(2) Para indicar que não existe espaços disponíveis, este campo pode manter o valor -1.
(3) Assim, cada registro removido, deve ter um marcado de que foi removido (e.g., '*', '#', …) e o offset para o próximo espaço disponível (ou -1 se não existem mais espaços disponíveis).
    a. Como sugestão pode-se usar a seguinte organização para os registros removidos: <tamanho em bytes do registro removido>*<offset para o próximo elemento da lista>, onde '*' é um marcador indicando que este espaço está disponível
(4) um novo espaço disponível deve ser acrescentado sempre no início da lista. Logo, deve-se atualizar o offset do cabeçalho e guardar o seu antigo offset no novo elemento da lista.

3 - Compactação

A estratégia de remoção vai criar fragmentos (internos e externos). Essa funcionalidade deve reconstruir o arquivo, quando solicitado pelo usuário, compactando todos os registros e limpando esses fragmentos (internos e externos). Isto é, ao fim da compactação, o arquivo deve ter apenas os registros validos e nenhum caracter invalido entre eles.

4 - Carrega Arquivos

A fim de facilitar os testes e avaliação do projeto, serão fornecidos dois arquivos:

a) "insere.bin"
b) "remove.bin"

O arquivo (a) conterá os dados a serem inseridos durante os testes. Não necessariamente todos os dados serão inseridos, isto é, está funcionalidade deve perguntar ao usuário qual registro deve ser inserido. Para tanto, uma sugestão é carregar o arquivo em memória (um vetor de struct, por exemplo) e ir acessando cada posição conforme as inserções vão ocorrendo.

O arquivo (b) conterá uma lista de chaves, "CodCli+CodVei", a serem utilizados durante a remoção. A ideia é a mesma já descrita, ou seja, carregar o arquivo em memória (um vetor de struct, por exemplo) e ir acessando cada posição conforme as remoções vão ocorrendo.

Nesses arquivos os registros seguem uma organização de registro e campo de tamanho fixo (que difere da estrutura que deve ser usada no projeto). Ver códigos fonte fornecidos.

Observações:

(1) Não criar o arquivo toda vez que o programa for aberto (fazer verificação). Isto é, o programa pode ser encerrado, e ao recomeçar deve continuar com o arquivo do estado que parou!

(2) O arquivo deve ser manipulado totalmente em memória secundária!

(3) Criar um pequeno menu para acessar cada uma das funcionalidades (1, 2 e 3, sendo que a 4 deve ser ativada ao iniciar o programa). Note que as funcionalidades podem ser executadas de forma aleatório de acordo com a necessidade do usuário. Por exemplo, 3 inserções → 1 remoção → compactação → 1 remoção → 2 inserções.

(4) A avaliação terá uma dinâmica como o seguinte exemplo:

1. execute o programa
2. insira um registro – 3 (índice do arquivo insere.bin)
3. insira um registro – 5 (índice do arquivo insere.bin)
4. insira um registro – 1 (índice do arquivo insere.bin)
5. feche o programa
6. abra o arquivo no editor hexadecimal
7. execute o programa novamente
8. remova o registro – 2 (índice do arquivo remove.bin)
9. feche o programa
10. abra o arquivo no editor hexadecimal
11. execute o programa novamente
12. faça uma compactação
13. feche o programa
14. abra o arquivo no editor hexadecimal
15. …