Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Organização e Recuperação da Informação
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 01 - INDEXAÇÃO

Atenção

1. **Prazo de entrega: 13/10/2019 às 23h55 (submissão online)**
2. **Sistema de submissão:** `http://judge.sor.ufscar.br/ori/`
3. A atividade deverá ser realizada individualmente

Organização de arquivos e indexação

Encontrar carona sempre foi uma mão na roda para evitar o gasto (e transtorno) de pegar o busão (muitas vezes lotado) para ir e voltar da universidade. Com a crise e a grana curta, a quantidade de alunos interessados em carona vem aumentando consideravelmente nos últimos anos. Contudo, o sistema de oferta ainda é muito precário no campus. Pensando nisso, surgiu a ideia do `UFSCarona`, um sistema com interface simples e objetiva para que pessoas possam divulgar vagas nos seus trajetos de carro. Para promover a criação desse sistema inovador e promissor, você foi convocado para desenvolver um protótipo em C que permita ao usuário manter uma base de dados de ofertas de vagas. Para isso, será necessário armazenar os seguintes dados:

- *Código*: composição de letras maiúsculas da primeira letra do nome do motorista, seguida das três primeiras letras da placa do veículo, seguidas do dia e mês da data do trajeto (com dois dígitos cada) e a hora da partida (dois primeiros dígitos). Ex: `AFGZ240907`. Esse campo é a *chave primária*, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do motorista* (primeiro nome, pelo qual os caronistas entrarão em contato, ex: `ALEXANDRE`);
- *Gênero* (gênero do motorista, Masculino (`M`) ou Feminino (`F`));
- *Data de nascimento* (data no formato `DD/MM/AAAA`, ex: `24/09/1999`);
- *Celular* (número de contato do motorista com DDD, ex: `(15) 99815-1234`);
- *Veículo* (veículo que será usado no trajeto, ex: `BELINA`);
- *Placa* (placa do veículo, ex: `FGZ-1234`).

- *Trajeto* (campo multi-valorado separado pelo caractere '|'. O primeiro valor será sempre a origem e o último o destino, porém podem ter um ou mais trajetos no meio, ex: `AFONSO VERGUEIRO|PANNUNZIO|UFSCAR`);
- *Data do trajeto* (data no formato `DD/MM/AA`, ex: `24/09/19`);
- *Hora da partida* (hora no formato `HH:MM`, ex: `07:30`);
- *Valor* (preço da carona no formato `999.99`, ex: `004.50`);
- *Vagas* (quantidade de vagas disponíveis, ex: `3`).

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

Desenvolva um programa que permita ao usuário manter uma base de dados de oferta de `CARONAS`. O programa deverá:

1. Inserir uma nova carona;
2. Remover uma carona a partir da chave primária;
3. Modificar o campo **vagas** de uma carona a partir da chave primária;
4. Buscar caronas a partir:
   1) da chave primária;
   2) da data da viagem;
   3) de uma localidade; ou
   4) de uma localidade em uma data.
5. Listar todas as caronas ordenadas por:
   1) código;
   2) trajeto e chave primária;
   3) nome do motorista e chave primária;
   4) data e hora da viagem; ou
   5) trajeto e data/hora.
6. Liberar espaço.
7. Imprimir arquivo de dados.
8. Imprimir arquivo de índices secundários:
   1) Índice secundário para os nomes dos motoristas;
   2) Índice secundário para as datas;

3) Índice secundário para os horários;

4) Índice secundário para os trajetos.

9. Finalizar (liberar estruturas da memória RAM).

Para realizar essa tarefa será necessário organizar e manter pelo menos um arquivo e cinco índices distintos:

(a) um arquivo de dados que conterá todos os registros;

(b) um índice primário;

(c) um índice secundário para os nomes dos motoristas;

(d) um índice secundário para os trajetos;

(e) um índice secundário para as datas; e

(f) um índice secundário para os horários.

Arquivo de dados

Como este trabalho será corrigido automaticamente por um juiz online que não aceita funções que manipulam arquivos, os registros serão armazenados e manipulados em *strings* que simularão os arquivos abertos. Você deve utilizar a variável global `ARQUIVO` e funções de leitura e escrita em *strings*, como `sprintf` e `sscanf`, para simular as operações de leitura e escrita em arquivo.

O arquivo de dados deve ser ASCII (arquivo texto), organizado em registros de tamanho fixo de 256 bytes (256 caracteres). Os campos *nome do motorista*, *modelo do carro* e *trajeto* devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: *código* (10 bytes), *gênero* (1 byte), *data de nascimento* (10 bytes), *celular* (15 bytes), *placa do veículo* (8 bytes), *data* (8 bytes) e *hora* (5 bytes) da carona, *valor* (6 bytes) e por fim *vagas disponíveis* (1 byte). A soma de bytes dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 256 bytes. Os campos do registro devem ser separados pelo caractere delimitador `@` (arroba). Cada registro terá 12 delimitadores, mais 64 bytes ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 180 bytes. Caso o registro tenha menos de 256 bytes, o espaço adicional deve ser marcado com o caractere ♯ de forma a completar os 256 bytes. Para evitar que o registro exceda 256 bytes, os campos variáveis devem ocupar no máximo 180 bytes. Exemplo:

```
ALEXANDRE@M@24/09/1999@(15) 99815-1234@BELINA@FGZ-1234@AFONSO
VERGUEIRO|PANNUNZIO|UFSCAR@24/09/19@07:30@004.50@3@###########
############################################################
############################################################
########GABRIEL AUGUSTO@M@22/07/1995@(11) 99542-4321@CORSA@ABC
-4321@SOROCABA|SAO ROQUE|COTIA|SAO PAULO@30/10/19@15:00@020.00
@4@##########################################################
############################################################
###############LETICIA FERREIRA DA SILVA@F@14/04/1990@(16) 954
35-2134@ONIX@CAC-1010@SAO PAULO|SAO JOSE DOS CAMPOS|TAUBATE|LO
RENA|VOLTA REDONDA|RIO DE JANEIRO@12/10/19@07:00@150.00@3@####
############################################################
########################ROBERT DOWNEY JUNIOR F :(@M@04/04/1965@
(11) 98754-1252@E-TRON GT@MAN-6969@LAS VEGAS|LOS ANGELES@25/04
/20@10:15@599.90@1@#############################################
############################################################
################################
```

Note que não há quebras de linhas no arquivo (elas foram inseridas aqui apenas para exemplificar a sequência de registros).

Instruções para as operações com os registros:

- **Inserção:** cada carona deverá ser inserida no final do arquivo de dados e atualizada nos índices.
- **Remoção:** o registro deverá ser localizado acessando o índice primário. A remoção deverá colocar os caracteres `*|` nas primeiras posições do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções. Observe que o registro deverá continuar ocupando exatamente 256 bytes. Além disso, no índice primário, o RRN correspondente ao registro removido deverá ser substituído por -1.
- **Atualização:** o único campo alterável é o de *Vagas*. O registro deverá ser localizado acessando o índice primário e a quantidade de vagas deverá ser atualizada no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo de *Vagas* sempre terá 1 dígitos.

Índices

Pelo menos, os seguintes índices deverão ser criados:

- `iprimary`: índice primário, contendo a chave primária e o RRN do respectivo registro, ordenado pela chave primária;
- `idriver`: índice secundário, contendo o nome do motorista e a chave primária do respectivo registro, ordenado pelo nome do motorista e em caso de empate, pelo código.
- `idate`: índice secundário, contendo a data e a chave primária do respectivo registro, ordenado pela data. Em caso de caronas com a mesma data, ordenar pelo código.

- `itime`: índice secundário, contendo o horário e o código da carona, deve ser ordenado primeiramente pela hora em ordem ascendente e, em seguida, pelo código.

- `iroute`: índice secundário, contendo a trajetória da carona e a chave primária do respectivo registro, ordenado pelos nomes dos trajetos e em seguida pelo código, esse índice deverá ser uma **lista invertida**.

Deverá ser desenvolvida uma rotina para a criação de cada índice. Os índices serão sempre criados e manipulados em memória principal na inicialização e liberados ao término do programa. Note que o ideal é que `iprimary` seja o primeiro a ser criado. Quanto aos índices secundários, `idriver`, `idate` e `itime` deverão ser **índices simples**, enquanto que `iroute` deverá ser uma **lista invertida**.

Para que isso funcione corretamente, o programa, ao iniciar precisa realizar os seguintes passos:

1. Perguntar ao usuário se ele deseja informar um arquivo de dados:
   - Se sim: recebe o arquivo inteiro e armazena no vetor `ARQUIVO`.
   - Se não: considere que o arquivo está vazio.
2. Inicializar as estruturas de dados dos índices.

Interação com o usuário

O programa deve permitir interação com o usuário pelo console/terminal (modo texto) via menu. As seguintes operações devem ser fornecidas (nessa ordem):

1. **Cadastro.** O usuário deve ser capaz de inserir uma nova carona. Seu programa deve ler os seguintes campos (nessa ordem): **nome do motorista**, **gênero**, **data de nascimento**, **celular**, **modelo do veículo**, **placa**, **trajeto**, **data da carona**, **hora**, **valor** e **quantidade de vagas**. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la nos índices. Garantidamente, os campos serão fornecidos de maneira regular, não sendo necessário um pré-processamento da entrada, exceto na opção de **Alteração**. Se um novo registro possuir a chave gerada igual a de um outro registro já presente no arquivo de dados, a seguinte mensagem de erro deverá ser impressa: "`ERRO: Já existe um registro com a chave primária AAAA999999.\n`", onde `AAAA999999` corresponde à chave primária do registro que está sendo inserido e `\n` indica um pulo de linha após a impressão da frase.

2. **Alteração.** O usuário deve ser capaz de alterar a quantidade de vagas de uma carona informando a sua chave primária. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (1 byte com o valor entre 0 e 9) e, nesse caso, altere o valor do campo diretamente no arquivo de dados. Caso contrário, exiba a mensagem "`Campo inválido!\n`" e solicite a digitação novamente.

3. **Remoção.** O usuário deve ser capaz de remover uma carona. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu. Para remover uma carona, seu programa deverá solicitar como entrada ao usuário somente o campo chave primária e a remoção deverá ser feita por um marcador.

4. **Busca.** O usuário deve ser capaz de buscar por uma carona:

   - **1. por código:** solicitar ao usuário a chave primária. Caso a carona não exista, seu programa deve exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu principal. Caso a carona exista, todos os seus dados devem ser impressos na tela de forma formatada, exibindo os campos na mesma ordem de inserção.
   - **2. por data:** solicitar ao usuário a data da viagem (formato `DD/MM/AA`). Caso não haja carona na data informada, o programa deve exibir a mensagem "`Registro não encontrado!\n`" e retornar ao menu principal. Caso existam uma ou mais ofertas de carona na mesma data, os registros completos deverão ser mostrados na tela de forma formatada, **ordenados pela chave primária** (ordem crescente).
   - **3. por localidade:** solicitar ao usuário uma localidade. Caso não tenha nenhuma carona cadastrada cujo trajeto passe pelo local informado, o programa deve exibir a mensagem "`Registro(s) não encontrado!\n`" e retornar ao menu principal. Caso existam uma ou mais caronas que passem pelo local informado pelo usuário, os registros completos desses deverão ser mostrados na tela de forma formatada, **ordenados pela chave primária** (ordem crescente).
   - **4. por localidade e data:** solicitar ao usuário uma localidade e uma data. Caso não tenha nenhuma carona cadastrada cujo trajeto passe pelo local na data informada, o programa deve exibir a mensagem "`Registro(s) não encontrado!\n`" e retornar ao menu principal. Caso existam uma ou mais caronas que passem pelo local informado na data definida pelo usuário, os registros completos desses deverão ser mostrados na tela de forma formatada, **ordenados pela chave primária** (ordem crescente).

5. **Listagem.** O sistema deverá oferecer as seguintes opções de listagem:

   - **1. por código:** exibe as caronas na ordem lexicográfica de código.
   - **2. por trajeto:** exibe todas as caronas ordenadas por trajeto e ordem lexicográfica do código.
   - **3. por nome do motorista:** exibe todas as caronas ordenadas pelo motorista (ordem lexicográfica do nome), seguida pela ordenação do código.
   - **4. por data e hora:** exibe as caronas em ordem crescente de data e hora. Em caso de empate, apresentar em ordem lexicográfica do código.
   - **5. por trajeto e data e hora:** exibe todas as caronas ordenadas por trajeto e ordem cronológica.

   As listagens deverão exibir todos os registros (exceto os marcados como excluídos) de maneira formatada e separados por uma linha vazia. Caso nenhum registro tenha sido apresentado, o programa deve exibir a mensagem "`Registro(s) não encontrado!\n`" e retornar ao menu.

6. **Liberar espaço.** O arquivo de dados deverá ser reorganizado com a remoção física de todos os registros marcados como excluídos e os índices deverão ser atualizados. A ordem dos registros no arquivo limpo não deverá ser diferente da original.

7. **Imprimir arquivo de dados.** Imprime o arquivo de dados. Caso esteja vazio, apresente a mensagem "`Arquivo Vazio!`"

8. **Imprimir arquivo de índices secundários.** O sistema deverá oferecer as seguintes opções de impressão:

   - **1. Índice de nomes:** imprime o arquivo de índice secundário para nomes de motoristas, o **idriver**. Caso o arquivo esteja vazio, imprimir "`Arquivo Vazio!`".
   - **2. Índice de datas:** imprime o arquivo de índice secundário para datas de caronas, o **idate**. Caso o arquivo esteja vazio, imprimir "`Arquivo Vazio!`".
   - **3. Índice de horário:** imprime o arquivo de índice secundário para os horários das caronas, o **itime**. Caso o arquivo esteja vazio, imprimir "`Arquivo Vazio!`".
   - **4. Índice de trajetos:** imprime o arquivo de índice secundário para os trajetos, o **iroute**. Caso o arquivo esteja vazio, imprimir "`Arquivo Vazio!`".

9. **Finalizar.** Libera memória e encerra a execução do programa.

Implementação

Implementar rotinas que contenham obrigatoriamente as seguintes funcionalidades:

- Estruturas de dados adequadas para armazenar os índices na memória principal;
- Verificar se o arquivo de dados existe;
- Criar o índice primário: deve refazer o índice primário a partir do arquivo de dados;
- Criar os índices secundários: deve refazer os índices secundários a partir do arquivo de dados;
- Inserir um registro: modificando o arquivo de dados e os índices na memória principal.
- Buscar por registros: busca pela chave primária ou por uma das chaves secundárias.
- Alterar um registro: modificando o campo do registro diretamente no arquivo de dados.
- Remover um registro: modificando o arquivo de dados e o índice primário na memória principal.
- Listar registros: listar todos os registros ordenados pela chave primária ou por uma das chaves secundárias.
- Liberar espaço: organizar o arquivo de dados e refazer os índices.

Dicas

- Ao ler uma entrada, tome cuidado com caracteres de quebra de linha (\n) não capturados.
- Você nunca deve perder a referência do começo do arquivo, então não é recomendável percorrer a *string* diretamente pelo ponteiro `ARQUIVO`. Um comando equivalente a `fseek(f, 192, SEEK_SET)` é `char *p = ARQUIVO + 192`.
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura.
- O `sprintf` adiciona automaticamente o caractere `\0` no final da *string* escrita. Em alguns casos você precisará sobrescrever a posição manualmente. Você também pode utilizar o comando `strncpy` para escrever em strings, esse comando, diferentemente do `sprintf`, não adiciona o caractere nulo no final.
- A função `strtok` permite navegar nas substrings de uma certa string dado o(s) delimitador(es). Porém, tenha em mente que ela deve ser usada em uma cópia da string original, pois ela modifica o primeiro argumento.
- É permitido (e recomendado) utilizar as funções de `qsort` e `bsearch`. Leia atentamente a documentação antes de usá-las.
- Para o funcionamento ideal do seu programa, é necessário utilizar a busca binária, porém tenha em mente que ela não garante que irá retornar o primeiro elemento do tipo, caso haja repetição.
- Caso a sua submissão esteja retornando funções restringidas é possível que a sua implementação possua funções que estejam acessando memória indevida.

CUIDADOS

Leia atentamente os itens a seguir.

O projeto deverá ser enviado pelo sistema online observando as seguintes regras:

1. Submeter um único arquivo `.c` chamado `{RA}_ORI_T01.c`, ex: `123456_ORI_T01.c`
2. Utilizar a linguagem `ANSI C`;
3. **Identificadores de variáveis:** escolha nomes apropriados;
4. **Documentação:** inclua cabeçalho, comentários e indentação no programa;
5. **Dificuldades em implementação:** consultar os monitores da disciplina nos horários estabelecidos;
6. **Erros de compilação:** nota **zero** no trabalho;

7. **Tentativa de fraude:** nota **zero na média** para todos envolvidos. Enfatizo que a detecção de cópia de parte ou de todo código-fonte, de qualquer origem, implicará na reprovação direta na disciplina. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s) devem ser devidamente documentadas em comentários no referido trecho. O que **NÃO** autoriza a cópia de trechos de código nem a codificação em conjunto. Portanto, compartilhem ideias, soluções, modos de resolver o problema, mas não o código.