Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Estruturas de Dados II
Prof. Tiago A. Almeida <talmeida@ufscar.br>

TRABALHO 01 - INDEXAÇÃO

Atenção

- **Prazo de entrega: 05/10/2016 – 23h55 (via Moodle).**
  Após o horário limite, o trabalho não será considerado!
- **A atividade deverá ser realizada individualmente.**

Organização de arquivos e indexação

Pokémon GO é um jogo *mobile* de realidade aumentada, onde o objetivo do jogador é encontrar, capturar, batalhar e treinar as criaturas virtuais que dão nome à aplicação - os *Pokémons*. Usando o GPS e a câmera do *smartphone*, o aplicativo espalhou *Pokémons* virtuais pelo mundo todo, *PokéStops* (locais onde o jogador pode recolher itens, como *Pokébolas* e ovos) e Ginásios (locais a serem disputados pelos jogadores que já pertencem a uma das três equipes do jogo).

Apesar de recente, *Pokémon GO* já conquistou seu lugar entre os aplicativos mais usados no mundo e chegou a se tornar o jogo *mobile* mais popular dos Estados Unidos em apenas cinco dias após seu lançamento. Todo esse sucesso pode ser comprovado pelas notícias sobre grandes conglomerações de pessoas em praças e outros locais públicos, atraídas por algum *Pokémon* que "apareceu" por lá.

Um grupo de jogadores de *Pokémon GO* resolveu registrar os *Pokémons* dos treinadores de cada equipe para saber qual o "arsenal" que cada uma dispõe em sua cidade e escolher a equipe mais bem "preparada". Você foi o escolhido para implementar esse sistema e deverá, para cada *Pokémon*, armazenar os seguintes dados:

- *Código* (composição de letras maiúsculas da primeira letra do nome da equipe, seguida da primeira letra do nome do treinador, seguida das duas primeiras letras do nome do *Pokémon* e do dia, mês, hora e minuto da captura (com dois dígitos cada)). Ex: `MABU13081125` - esse campo é a chave primária, portanto, não poderá existir outro valor idêntico na base de dados;
- *Nome do* Pokémon (nome da espécie de *Pokémon*. Ex: `Bulbasaur`);
- *Tipo(s) do* Pokémon (separados por uma barra '/', caso o *Pokémon* se enquadre em mais de um. Ex: `Grama/Venenoso`);
- *Combat Points – CP* (pontos de combate do *Pokémon*, com 4 dígitos inteiros e 2 dígitos decimais, separados por um ponto `'.'`. Ex: `1013.06`);

- *Data de captura* (data de captura do *Pokémon*, no formato `DD/MM/AA`. Ex: `13/08/16`);
- *Hora de captura* (hora da captura do *Pokémon*, no formato `HH:MM`. Ex: `11:25`);
- *Nome do treinador* (nome do treinador que capturou o *Pokémon*. Ex: `Ash`);
- *Nível do treinador* (nível do treinador que capturou o *Pokémon*, com 3 dígitos inteiros. Ex: `095`);
- *Nome da equipe* (nome da equipe do treinador que capturou o *Pokémon*. Ex: `Mystic`);

Garantidamente, nenhum campo de texto receberá caractere acentuado.

Tarefa

Desenvolva um programa que permita ao usuário manter uma base de dados de *Pokémons*. O programa deverá permitir:

1. Inserir um novo *Pokémon*;
2. Remover um *Pokémon* a partir da chave primária;
3. Modificar o campo *combat points* de um *Pokémon* a partir da chave primária;
4. Buscar *Pokémons* a partir:
   1) da chave primária,
   2) do nome do *Pokémon*, ou
   3) do nome da equipe.
5. Listar todos os *Pokémons* da base ordenados por:
   1) código (ordem lexicográfica),
   2) nome do *Pokémon* (ordem lexicográfica), ou
   3) nome da equipe (ordem lexicográfica).
6. Liberar espaço.

Para realizar essa tarefa será necessário organizar quatro arquivos distintos: (a) um arquivo de dados que conterá todos os registros, (b) um arquivo de índice primário, (c) um arquivo de índice secundário para o nome do *Pokémon* e (d) um arquivo de índice secundário para o nome da equipe.

Arquivo de dados

O arquivo de dados deve ser ASCII (arquivo texto) e organizado em registros de tamanho fixo de 192 *bytes*. Os campos que possuam letras em sua composição devem ser convertidos para que todas elas sejam maiúsculas no arquivo de dados. Os campos `nome do Pokémon`, `tipo do Pokémon`, `nome do treinador` e `nome da equipe` devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: `combat points` (7 *bytes*), `data de captura` (8 *bytes*), `hora da captura` (5

bytes), `nível do treinador` (3 *bytes*) e `código` (*chave primária* - 12 *bytes*). A soma de *bytes* dos campos fornecidos (incluindo os delimitadores necessários) nunca poderá ultrapassar 192 *bytes*. Os campos do registro devem ser separados pelo caractere delimitador `@` (arroba). Cada registro terá 9 delimitadores, mais 35 *bytes* ocupados pelos campos de tamanho fixo. Você precisará garantir que os demais campos juntos ocupem um máximo de 148 *bytes*. Caso o registro tenha menos de 192 *bytes*, o espaço adicional deve ser marcado com o caractere ♯ de forma a completar os 192 *bytes*. Para evitar que o registro exceda 192 *bytes*, cada campo de tamanho variável deve ocupar no máximo 37 *bytes*. O programa deve impedir a inserção de registros com campos de mais de 37 *bytes*.

```
MABU13081125@BULBASAUR@GRAMA/VENENOSO@1013.06@13/08/16@11:25@ASH
@095@MYSTIC@###############################################################
###############################################################
IBGE10081309@GEODUDE@PEDRA/TERRA@0808.00@10/08/16@13:09@BROCK@08
8@INSTINCT@###############################################################
###############################################################
VMPS26082211@PSYDUCK@AGUA@1010.55@26/07/16@2211@MISTY@093@VALOR@
###############################################################
###############################################################
MAPI08080500@PIKACHU@ELETRICO@0876.54@08/08/16@05:00@ASH@095@MYS
TIC@###############################################################
###############################################################
MACH02090259@CHARIZARD@FOGO/VOADOR@2222.22@02/09/16@02:59@ASH@09
5@MYSTIC@###############################################################
###############################################################
```

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

O arquivo de dados não deverá conter cabeçalho e deverá se chamar `pokemons.dat`.

Instruções para as operações com os registros:

- **Inserção:** cada *Pokémon* deverá ser inserido no final do arquivo de dados e atualizado 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 192 *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 `combat points`. O registro deverá ser localizado acessando o índice primário e a pontuação deverá ser atualizada no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo `combat points` sempre terá 7 dígitos.

Índices

Três arquivos com índices deverão ser criados:

- `iprimary.idx`: índice primário, contendo a chave primária e o RRN do respectivo registro, ordenado pela chave primária.
- `ipokemon.idx`: índice secundário, contendo o nome do *Pokémon* e a chave primária do respectivo registro, ordenado pelo nome do *Pokémon*.
- `iteam.idx`: índice secundário, contendo o nome da equipe do treinador que capturou o *Pokémon* e a chave primária do respectivo registro, ordenado pelo nome da equipe.

Deverá ser criada uma rotina para a criação de cada índice. Os índices serão criados carregando para a memória principal os dados necessários, procedendo a ordenação em memória principal e a seguir gravando o arquivo de índice ordenado. Note que o ideal é que `iprimary.idx` seja o primeiro a ser criado.

Ao iniciar o programa, esse deverá carregar os índices para a memória principal e durante a execução do programa, manipular os índices na RAM. Ao fechar o programa, os arquivos de índices deverão ser atualizados no disco.

Para que isso funcione corretamente, o programa, ao iniciar deverá:

1. Verificar se existe o arquivo de dados.
   - Se existir: abrir para escrita e leitura e passar para o item 2.
   - Se não existir: criar o arquivo de dados no disco, abrindo para escrita e leitura.
2. Verificar se existem os arquivos de índices:
   - Se existirem: verificar se estão consistentes com o arquivo de dados (usar uma *flag* para isso).
     i Se estiverem consistentes: carregar os índices para a RAM.
     ii Senão: refazer os índices na RAM e gravá-los no disco.
   - Se não existirem: criar os índices na RAM e gravá-los no disco.

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 um novo *Pokémon*. Seu programa deve ler os seguintes campos (nessa ordem): `nome do Pokémon`, `tipo do Pokémon`, `combat points`, `data de captura`, `hora da captura`, `nome do treinador`, `nível do treinador` e `nome da equipe`. Note que a chave **não é** inserida pelo usuário, você precisa gerar a chave para gravá-la no registro. Antes de armazenar um campo, você deve fazer algumas verificações:
   - O `nome do Pokémon` informado deve ser composto apenas por letras.

- O `tipo do Pokémon` informado deve ser composto apenas por letras e, caso haja mais de um tipo a ser informado, deve ser composto apenas por letras e pelo separador '/' (um tipo não deve iniciar ou terminar com '/').
- Os `combat points` informados devem ser compostos por 7 *bytes* e estar no formato `NNNN.NN`, onde N pertence a [0, 9] e o separador corresponde a '.';
- A `data de captura` informada deve estar no formato "`DD/MM/AA`", onde `AA` é maior ou igual a 16 (referente ao ano corrente e de lançamento do jogo), `MM` pertence ao intervalo [1, 12], `DD` pertence ao intervalo [1, 31], [1, 30], [1, 29] ou [1, 28], de acordo com o mês e o ano, e os dois separadores correspondem a '/';
- A `hora da captura` informada deve estar no formato `HH:MM`, onde `HH` pertence ao intervalo [0, 23], `MM` pertence ao intervalo [0, 59] e o separador corresponde a ':';
- O `nível do treinador` informado deve ser composto por 3 *bytes* e pertencer ao intervalo [1, 100];
- O `nome da equipe` informado deve sempre corresponder a um dos três nomes a seguir: `Valor`, `Instinct` ou `Mystic` (como a entrada não tem padrão, recomenda-se que seja feita a conversão das letras para somente maiúsculas ou somente minúsculas antes de fazer a comparação).

Caso algum dos campos esteja irregular, exibir a mensagem "`Campo inválido! Informe novamente: `" e solicitar a digitação novamente. Se a chave primária gerada já existir no arquivo de dados, a seguinte mensagem de erro deverá ser impressa: "`ERRO: Já existe um registro com a chave primária AAAA99999999.\n`", onde `AAAA99999999` corresponde à chave primária do registro que está sendo inserido.

2. **Alteração.** O usuário deve ser capaz de alterar os pontos de combate de um *Pokémon*, informando a sua chave primária. Caso ela não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!`"e retornar ao menu. Caso o registro seja encontrado, certifique-se de que o novo valor informado está dentro dos padrões (7 *bytes*, no formato `NNNN.NN`, com `N` pertencente ao intervalo [1, 9] e um ponto '.' como separador) e, nesse caso, altere o valor do campo no arquivo de dados. Caso contrário, exiba a mensagem "`Campo inválido!`" e solicite a digitação novamente.

3. **Remoção.** O usuário deve ser capaz de remover um *Pokémon*. Caso ele não exista, seu programa deverá exibir a mensagem "`Registro não encontrado!`" e retornar ao menu. Para remover um *Pokémon*, 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 um *Pokémon*:

   - **1. por código:** solicitar ao usuário a chave primária. Caso o *Pokémon* não exista, seu programa deve exibir a mensagem "`Registro não encontrado!`" e retornar ao menu principal. Caso o *Pokémon* exista, todos os seus dados devem ser impressos na tela de forma formatada, exibindo os campos na mesma ordem de inserção.

- **2. por nome do *Pokémon*:** solicitar ao usuário o nome do *Pokémon*. Caso o *Pokémon* informado não tiver sido capturado por nenhum jogador, o programa deve exibir a mensagem "`Registro não encontrado!`" e retornar ao menu principal. Caso existam um ou mais *Pokémons* cujo nome seja o informado pelo usuário, os registros completos desses *Pokémons* deverão ser mostrados na tela de forma formatada, ordenados pela chave primária e separados por uma linha vazia.
- **3. por nome da equipe:** solicitar ao usuário o nome da equipe. Caso a equipe não tenha capturado nenhum *Pokémon*, o programa deve exibir a mensagem "`Registro não encontrado!`" e retornar ao menu principal. Caso existam um ou mais *Pokémons* que pertençam à equipe informada, os registros completos desses *Pokémons* deverão ser mostrados na tela de forma formatada, ordenados pela chave primária e separados por uma linha vazia.

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

   - **1. por código:** exibe os *Pokémons* na ordem lexicográfica de código.
   - **2. por nome do *Pokémon*:** exibe os *Pokémons* na ordem lexicográfica de seus nomes.
   - **3. por nome de equipe:** exibe os *Pokémons* na ordem lexicográfica de nome da equipe.

   A listagem deverá exibir todos os registros (exceto os marcados como excluídos) de maneira formatada, separados por uma linha vazia. Caso o arquivo de dados esteja vazio, o programa deve exibir a mensagem "`Arquivo vazio!`" 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.

7. **Finalizar.** Atualiza os índices no disco, fecha os arquivos, libera memória alocada e encerra a execução do programa.

Implementação

Implementar uma biblioteca de manipulação de arquivos para o seu programa, contendo obrigatoriamente as seguintes funcionalidades:

- Uma estrutura de dados para armazenar os índices na memória principal;
- Verificar se o arquivo de dados existe;
- Verificar se o índice primário existe;
- Verificar se os índices secundários existem;
- Criar o índice primário: deve refazer o índice primário a partir do arquivo de dados e substituir, caso haja, o índice existente no disco;
- Criar os índices secundários: deve refazer os índices secundários a partir do arquivo de dados e substituir, caso haja, os índices existentes no disco;

- Carregar os índices do disco para a memória principal;
- Inserir um registro: modificando o arquivo de dados no disco, e os índices na memória principal.
- Buscar por registros: busca pela chave primária ou por uma das chaves secundárias.
- Alterar um registro: modificando o arquivo de dados no disco.
- Remover um registro: modificando o arquivo de dados no disco 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: removendo fisicamente do arquivo de dados todos os registros marcados como excluídos, e atualizando os índices.
- Atualizar todos os índices: deverá ser chamada ao finalizar o programa e deverá gravar os arquivos de índices no disco a partir das estruturas da memória principal.

Utilizar a linguagem `ANSI C`. Separar a interface da implementação (arquivos `c` e `h`), e fornecer um `Makefile` para compilar o código e retornar ao estado inicial.

A implementação do seu trabalho deverá **obrigatoriamente** ser composto por três arquivos: uma rotina principal chamada `main.c`, um arquivo com as implementações de todas as funções chamado `{RA}_ED2_T01.c`, cujas definições deverão estar no arquivo `{RA}_ED2_T01.h`.

CUIDADOS

Leia atentamente os itens a seguir.

1. O projeto deverá ser enviado pelo Moodle observando a seguinte regra de nomenclatura:
   - Submeter um arquivo `zip` chamado `{RA}_ED2_T01.zip`. Tal arquivo deverá conter uma pasta chamada `{RA}_ED2_T01` com todos os arquivos que compõem o programa e o `Makefile`.
2. Não utilize acentos nos nomes de arquivos;
3. **Identificadores de variáveis:** escolha nomes apropriados;
4. **Documentação:** inclua cabeçalho, comentários e indentação no programa;
5. **Erros de compilação:** nota **zero** no trabalho;
6. **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 **idéias** 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 idéias, soluções, modos de resolver o problema, mas não o código.