Lucas Bertolino Pizzo – 220484

Este catalogador utiliza uma série de Arquivos Sequenciais Indexados de diferentes campos em seu funcionamento. O primeiro passo realizado pelo programa é executar uma série de funções que geram tais ASIs, começando pelo que utiliza o nome do artigo como chave, pois esta é a chave primária e as demais funções dependem de seu resultado.  
  
**catalogNames** abre o arquivo Artigos.csv e varre-o linha-a-linha, procurando pelo campo em que se encontra o nome do artigo, copiando esse nome para uma string e adicionando em uma nova linha na mesma string o endereço da linha em que se encontra em forma da sua distância ao início do arquivo Artigos.csv em bytes, daqui em diante referido como displacement. Cada string é adicionada a um vetor, que é ordenado assim que todos os nomes são processados. O vetor é então escrito no arquivo namesCatalog.bin, contendo o nome de cada artigo em ordem alfabética, junto com o displacement do arquivo principal.  
  
**catalogAuthors** busca os autores de cada artigo de maneira similar e os adiciona (e o displacement da linha em que foram encontrados) a um vetor, que também é ordenado e escrito em um arquivo – authorsCatalog. Este arquivo possui apenas um displacement por string, então catalogAuthors2 cria outro arquivo, authorsCatalog2, que coloca todos os displacements de um autor em uma só linha: estas duas etapas utilizam menos memória princpal do que um processo que gera diretamente um arquivo sem redundâncias necessitaria.  
Note que catalogAuthors utiliza os displacements de namesCatalog – desta maneira os displacements de cada autor já estão em ordem alfabética. catalogAuthors2 por sua vez traduz o displacement de namesCatalog para o displacement equivalente de Artigos.csv, já que tudo já está em ordem e o programa irá consultar todas as informações de cada artigo. **catalogCountries** e **catalogCountries2** funcionam de maneira praticamente idêntica a essas duas funções, assim como **catalogFields** e **catalogFields2**.  
  
**getName** varre namesCatalog por cada palavra de cada string e retorna todos os artigos que tiverem uma palavra idêntica à consultada em seu nome. **getAuthor** e **getCountry** procuram em seus respectivos catálogos por uma string idêntica à consultada, e retorna todos os artigos apontados pelos displacement associados a tal string.  
  
**getFieldCount** cria um vetor de estruturas Author com todos os autores de authorsCatalog2 e o número de artigos dessa área por esse autor, que é incrementado sempre que um displacement da área, indexado em fieldsCatalog2, é idêntico a algum displacement de um autor (novamente, indexado em authorsCatalog2). O vetor de Authors é então ordenado pelo contador decrescentemente e impresso na tela.  
  
**getAuthors** funciona de maneira similar a getAuthor e getCountry, porém necessita de duas ou mais entradas e executa o equivalente a um AND entre os displacements de um autor e o conjunto de displacements já processados. Novamente, esses displacements são consultados via catalogAuthors2.  
  
  
Todo o código não pertencente às bibliotecas padrão é original. A linha  
  
bool sortByCount(const Author &lhs, const Author &rhs) { return lhs.count < rhs.count; }  
  
foi adaptada de uma thread do site Stack Overflow sobre o uso de parâmetros customizados para o sort() padrão de C++.  
  
Todas as telas do programa informam comandos e entradas esperados. O programa apenas necessita que Artigos.csv esteja no mesmo diretório, e que siga estritamente a formatação usada pelo arquivo exemplo.  
  
A estrutura de dados utilizada neste trabalho foi Arquivo Sequencial Indexado. Devido ao tamanho variável das linhas de cada artigo e à maneira que sobrescrita de arquivos funciona, sorts baseados em disco, bem como a organização dos ASIs em árvores seriam inviáveis, pois precisariam de muito *padding* (advinhado) para garantir que todos os elementos tivessem o mesmo tamanho e portanto que sobreescritas não destruíssem a integridade do arquivo.