Organização de Arquivos

Trabalho Prático (Parte 1)

Disciplina: SCC 215 - Organização de Arquivos

Curso: Bacharelado em Ciências de Computação

Docente: Professora Doutora Cristina Dutra de Aguiar Ciferri

Integrantes do grupo	**Nº USP**
Caroline Jesuíno Nunes da Silva	9293925
João Victor Garcia Coelho	9845221
Matheus Sanchez	9081453
Pedro Paulo Isnard Brando	9845221

Data de entrega: 09/05/2018

Índice

Seção 1

Nesta seção será abordada a implementação da funcionalidade um, que consistia na leitura de um arquivo de registros no formato CSV e gravação dos registros lidos num arquivo de saída.

Para realizar esta funcionalidade, um os tipos Arquivo e Registro foram definidos no programa:

- O tipo Arquivo possui o número de registros lidos pelo programa, para que tivéssemos esse controle; o status do arquivo e o topo da pilha que indica os removidos, que devem estar presentes no cabeçalho do arquivo de saída; e um vetor de registros lidos, que define o arquivo em si e é do tipo Registro.
- O tipo Registro possui o código de cada escola (int); a data de início do ano letivo (char[11]); a data de final do ano letivo (char[11]); indicador de tamanho do campo do nome da escola (int); string com o tamanho do nome da escola (char *); indicador de tamanho do campo do município (int); string com o nome do município (char *); indicador de tamanho do campo do endereço (int); string com o endereço (char *).

O nome do arquivo de entrada é lido pela linha de execução.

É chamada a função '*le_dados()*' com o nome do arquivo como parâmetro. O arquivo é aberto em modo de leitura, seu tamanho é contado e inicializa-se um novo Arquivo. Enquanto o arquivo não chega ao fim, aloca-se um novo registro no vetor contido no arquivo novo, lê-se o código da escola do arquivo de origem e atribui-se o valor lido ao campo código do registro atual. Em seguida, lê-se os indicadores dos tamanhos de todos os campos de variáveis e esses campos, utilizando a função auxiliar '*le_tamanho_variável()*' que lê os itens de qualquer tamanho desde que não haja um delimitador no meio.

Após essa leitura, é usada uma leitura auxiliar de um caractere para verificar se ele é um delimitador. Se ele for, significa que a data é vazia, e a string definida como substituta para datas vazias é atribuída para sua posição ('*0000000000\0*'). Caso contrário, o caractere lido é colocado na primeira posição da string de data e o resto da data é lido e definido, e o '*\0*' é acrescentado no fim da string. O mesmo processo é repetido para a data de final.

Ao fim desse processo, o número de registros lidos é incrementado, e a estrutura do Arquivo é retornada.

Com essa estrutura que foi retornada, é chamada a função '*arquivo_saida()*' que recebe como parâmetro o endereço da estrutura Arquivo. Essa função abre um arquivo de saída chamado "*saida.bin*" e, registro a registro, todos os arquivos são escritos no arquivo em binário e, caso o tamanho do registro seja menor que o tamanho destinado a ele, o resto dos bits é preenchido .

Seção 2

Nesta seção será abordada a implementação da funcionalidade dois, que permite a recuperação dos dados de todos os registros e os mostra de forma organizada, com distinção de campos, na saída padrão.

Para a execução dessa funcionalidade chama-se a função '*exibe_registros()*', que não recebe nenhum parâmetro. Dentro dela, abre-se o arquivo de saída no modo de leitura e conta-se o tamanho dele. Enquanto o arquivo não acaba, então, é chamada para cada registro a função auxiliar '*ImprimeRegistro()*', que recebe como parâmetro o ponteiro e o byte inicial.

O ponteiro é então colocado na posição correta e os campos do registro são lidos um a um e armazenados em variáveis auxiliares. Todas as variáveis são então impressas na saída padrão, tendo cada um dos campos separados por um espaço.

Seção 3

Nesta seção será abordada a implementação da funcionalidade três, que permite a recuperação dos dados de registros específicos que correspondam a um filtro determinado.

Para esta funcionalidade, o campo do filtro e o filtro são lidos pela linha de execução. Abre-se o arquivo de saída em modo de leitura e chama-se a função '*func3()*' com o ponteiro do arquivo de saída, o nome do campo do filtro e o filtro como parâmetros.

Nessa função é verificado se o nome do campo do filtro é de um dos campos de tamanho fixo para que possamos definir diretamente qual o *byte offset* correto para efetuar a busca em cada registro no arquivo de saída. Nesse caso, pulam-se os bytes do cabeçalho e, até o fim do arquivo, soma-se o *byte offset* encontrado ao ponteiro do arquivo e realiza-se a leitura do dado na posição atual: se o campo for o código da escola, lê-se como inteiro; caso contrário, como string. Neste caso, realiza-se um '*strcmp()*' entre a string dada e a string lida e, se elas forem iguais, chama-se a função '*ImprimeRegistro()*'; naquele, compara-se os números e, se forem o mesmo, a mesma função é chamada. No fim desse processo, soma-se ao ponteiro, o tamanho do registro para avançar para o mesmo campo na próxima linha.

Nos casos em que o campo pedido não possui tamanho fixo, verifica-se qual é o campo pedido. Soma-se ao ponteiro de arquivo o tamanho dos campos iniciais e o tamanho do cabeçalho, para que ele já fique na posição correta para começar a procurar. De forma geral, para cada registro pertencente a esses campos, lê-se inicialmente o tamanho do campo e, se for o especificado, lê a string e a compara: se for uma combinação, o registro é impresso; caso contrário, soma-se ao ponteiro o tamanho do campo lido e repete-se o processo até chegar no campo especificado: a partir desse ponto, o processo de identificação se repete.

Seção 4

Nesta seção será abordada a implementação da funcionalidade quatro, que permite a recuperação dos dados de um determinado RRN, especificado pelo usuário.

Para a realização dessa funcionalidade, o RRN é lido pela linha de execução e o arquivo de saída é aberto. A função '*ImprimeRegistro()*' já supracitada e explicada é chamada com o ponteiro do arquivo de saída e o tamanho do registro multiplicado pelo RRN lido e somado ao tamanho do cabeçalho como parâmetros.

Seção 5

Nesta seção será abordada a implementação da funcionalidade cinco, que consiste na remoção lógica de registros no arquivo de saída.

Para essa funcionalidade, o RRN é lido via comando de execução e abre-se o arquivo de saída no modo de leitura e escrita. Chama-se então a função '*RemoveRegistro()*', com o ponteiro do arquivo e o RRN lido como parâmetros.

Nessa função verifica-se se o arquivo possui tamanho suficiente para armazenar o RRN informado: em caso negativo, retorna-se; em caso positivo, pula-se o status do arquivo, lê-se o topo da pilha, soma-se ao ponteiro de arquivo a multiplicação do RRN pelo tamanho do registro mais o tamanho do cabeçalho e verifica se o registro já foi removido. Se sim, apenas retorna. Se não, escreve o indicador de remoção no registro, escreve o topo da pilha lido na posição seguinte, move o ponteiro de arquivo para o início, soma-se um para pular o byte indicador e escreve o RRN como topo da pilha.

Seção 6

Nesta seção será abordada a implementação da funcionalidade seis, que consiste na inserção de registros adicionais utilizando uma abordagem dinâmica de reaproveitamento de espaços.

A leitura dos dados a serem inseridos é dada pela linha de execução. O arquivo de saída é aberto em modo de leitura e escrita e chama-se a função de '*Insercao()*', tendo como argumentos o ponteiro do arquivo, o código da escola, e os outros valores lidos.

Nessa função, cria-se um novo registro e atribui-se a ele todos os valores lidos.

Verifica-se então se o RRN do local de inserção é -1. Se for, o registro deve ser inserido no fim do arquivo. Então calcula-se o tamanho do arquivo, multiplica-se esse número pelo tamanho de cada registro e soma-se ao resultado o tamanho do cabeçalho. Soma-se esse valor ao ponteiro de arquivo.

Caso contrário, soma-se então um ao ponteiro de arquivo para acessar o topo da pilha, multiplica-se o valor lido do topo pelo tamanho de cada registro, soma-se a esse valor o tamanho do cabeçalho[1]. O resultado dessa conta é somado ao ponteiro de arquivo que acessa agora o último registro removido. Lê-se o primeiro byte que é o indicador de remoção e salva-se o próximo valor. Volta-se o ponteiro para o começo do arquivo e soma-se um ao ponteiro para acessar a posição do topo da pilha. Imprime-se no arquivo o valor lido anteriormente. Soma-se novamente ao ponteiro de arquivo o valor calculado em 1.

Chama-se a função '*EscreveRegistro()*', com o ponteiro de arquivo, o registro criado e o RRN recuperado como parâmetros. Essa função escreve no arquivo na posição indicada os dados que estão no registro criado à parte. Em seguida, verifica se ainda há bytes sobrando. Se houver, completa o registro com zeros.

Seção 7

Nesta seção será abordada a implementação da funcionalidade sete, que permite a atualização de campos baseando-se em seu RRN.

Lê-se o RRN e os outros dados pela linha de execução, em seguida abre-se o arquivo de saída. Chama-se então a função '*updateRegistro()*', com o ponteiro de arquivo e os outros dados lidos como argumentos.

Dentro dessa função cria-se um registro com os dados a serem colocados na atualização e atribui-se a ele os dados passados como parâmetro. Verifica-se, em seguida, se o registro existe. Se sim, chama a função '*EscreveRegistro()*' já supracitada e explicada, com o ponteiro de arquivo de saída, o registro e o RRN lido como argumentos.

Seção 8

Nesta seção será abordada a implementação da funcionalidade oito, que corresponde à desfragmentação do arquivo de dados.

A função '*func8()*' é chamada. Abre-se o arquivo de saída e um outro arquivo em branco. Lê-se o cabeçalho do arquivo original e escreve-se o cabeçalho no novo arquivo.

Percorre-se o arquivo original até o final e, para cada registro existente, efetua a leitura e escreve o que foi lido no novo arquivo. No final da escrita de cada registro, soma ao ponteiro de arquivo o tamanho de registro para que avance para a próxima lista.

Seção 9

Nesta seção será abordada a implementação da funcionalidade nove, que permite a recuperação dos RRNs de registros removidos.

A função '*func9()*' é chamada. Ela abre o arquivo de saída, soma um ao ponteiro de arquivo para poder recuperar o valor do topo da pilha. Enquanto o valor que ela recuperar não for igual a -1, imprime-se o topo da pilha, multiplica esse valor pelo tamanho de registro e o soma ao tamanho do cabeçalho. Esse valor é então somado ao ponteiro de arquivo, que vai para o registro removido anteriormente. Pula-se o primeiro valor, que será o indicador de remoção e lê-se o próximo valor na variável que representa o topo da pilha.

Compilação e execução

Para compilar o programa basta navegar até a pasta em que todos os arquivos fonte se encontram e digitar no terminal o comando '*make*'.

Para rodar o programa digite no terminal '*./main*'.