Relatório TP1

Grupo 1
Anderson Phillip Birocchi (072787)
Miguel Francisco Alves de Mattos Gaiowski (076116)
Raphael Kubo da Costa (072201)

13 de março de 2008

Objetivo

Este trabalho prático tem como objetivo implementar um programa que seja capaz de coletar e armazenar dados sobre obras de arte e guardá-los em um arquivo ".dat" no disco rígido. Este deve ter sua estrutura dos dados padronizada, resultando num tamanho de arquivo múltiplo de 420 bytes.

Em específico, este trabalho apenas insere registros na base de dados de obras de arte.

Descrição do Trabalho

Definiu-se, inicialmente, dividir o sistema em vários arquivos de modo a facilitar sua manutenção e a escalabilidade. Apesar de o projeto apenas inserir dados na base, posteriormente seu uso crescerá e a divisão em arquivos em vários "módulos" facilita o planejamento e a engenharia do software.

O arquivo **data.h** é onde estão definidos as estruturas de dados usadas no programa. No futuro, quando o projeto crescer, mais tipos serão definidos ali.

Os arquivos **io.c** e **io.h** controlam a leitura e escrita de dados entrados pelo usuário. A função readData "conversa" com o usuário e chama a função readValues que lê os dados de forma apropriada. A funçao writeData toma conta da escrita dos dados no arquivo.

Convém, aqui, detalhar as funções *stripNewLine* e *readValue*. A validação das entradas do usuário é conhecidamente um problema da linguagem C, como se vê em *http://www.c-faq.com/stdio/index.html*. A função *readValue* recebe dois parâmetros: um *char* onde deve ser armazenada a entrada do usuário e um *size_t (unsigned int)* com seu tamanho máximo.

Optamos por usar a função *fgets* para a leitura de dados da entrada padrão, já que podemos especificar o tamanho da entrada. Funções da família *scanf* devem ser utilizadas com cautela apenas em entradas já formatadas. Se a entrada

do usuário fosse lida através dela, poderia-se, por exemplo, colocar um título com mais de 200 caracteres. Entretanto, o uso de *fgets* gera um inconveniente: caso o usuário entre com menos de *length* caracteres, o '\n' também é armazenado na string final. Para que ele seja eliminado quando existir, foi criada a função *stripNewLine*, que substiti o '\n' por um '\0'. Futuramente, pode-se aumentar sua utilidade removendo todos os caracteres especificados.

Além de *readValue*, foram criadas as funções *readInt* e *readString*, que utilizam *readValue*. Cada uma lê uma string ou um inteiro, checa se a entrada é nula e, no caso dos inteiros, se a entrada é realmente apenas numérica.

Ainda em **io.c**, existe também a função *flushBuffer*, que é usada quando uma entrada maior que a esperada é lida, e sobram caracteres no buffer de input. A função se encarrega de ler todos os caracteres até o EOF e descartá-los.

Os arquivos **menu.c** e **menu.h** carregam as funções de impressão de boas vindas e do menu de opções para o usuário. Como a impressão do menu é algo que ocorre com frequência, é saudável definir funções para tal. Novamente, outras funções úteis de I/O podem vir a ser acrescentadas no futuro, à medida que o sistema crescer.

O código contido em **main.c** contém a lógica de execução do programa, que deve imprimir o menu, sair caso esta seja a opção do usuário, ou inserir uma obra de arte perguntando, logo após, se deseja inserir outro registro.

Por enquanto, o loop de leitura da opção desejada pelo usuário está todo em **main.c**, mas, devido à escalabilidade quando novas opções forem inseridas, ele pode ser transferido para uma função à parte. Tentou-se validar a entrada do usuário da melhor forma possível, inclusive usando um *char input[2]*, com espaço para dois caracteres, para se ter certeza de que o usuário entrou com apenas uma letra. Como a leitura do menu não é case-sensitive, foi usada a função *tolower*, definida em **ctype.h** para transformar qualquer entrada em minúscula.

Resultado final

Conseguimos fazer um programa funcional, sem vazamentos de memória e sem bugs conhecidos. O ponto mais crítico nesta fase inicial do sistema foi a validação das entradas do usuário, conhecidamente um dos aspectos mais perigosos para programadores C novatos.

É feita uma checagem de entradas nulas e se dados inteiros contêm apenas dígitos. Entretanto, não é checada a presença de entradas repetidas ou de identificadores que correspondam a imagens inexistentes, visto que o trabalho para isso, pelo menos por enquanto, é grande e fora do escopo desta tarefa, por envolver a criação de estruturas de dados apenas para a busca de repetições e funções de manipulação de diretórios.