UNIVERSIDADE ESTADUAL DO CEARA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
Fortaleza, 24 de Novembro 2018

PROGRAMAÇÃO ESTR. E ORIENTADA A OBJETO
TRABALHO FINAL – RELATÓRIO

NABLA – um projeto para gerenciamento de escolas de dança

Silvio Eduardo Sales de Britto Ribeiro
Lucas Queiroz Frota
Isaac Honório Moreira

Orientações: Sidney Xavier Botão

1. A APLICAÇÃO

O software consiste em um sistema baseado em banco de dados com cadastro, lista e informações de alunos e professores de uma escola de dança. Um Menu com opções para o usuário é apresentado e, baseado nas escolhas e na interação com a aplicação, ela funciona.

As opções são:

*1. Cadastro de Aluno;

Nesta opção o usuário digita os dados referentes ao objeto previamente instanciado do tipo Aluno (*aluno1*, ou somente *aluno*) e o programa escreve diretamente em um arquivo *aluno.txt* o que for digitado enquanto o usuário solicitar. É usado *try-catch* para exibir uma mensagem de erro caso aconteça algo no momento do cadastro do aluno. Mostra a lista de alunos para conferir o cadastro.

*2. Remoção de Aluno;

Nesta opção é solicitado o nome do aluno que já esteja cadastrado no arquivo e o software se encarrega de sobrescrever os dados existentes no arquivo antigo, excluindo somente o aluno passado. Ao invés de ‘remover’ o programa ‘deixa de recadastrar’. Mostra a lista de alunos para conferir a remoção.

*3. Lista dos alunos cadastrados;

Mostra uma lista com os dados separados por espaço dos alunos que estão cadastrados no arquivo. É usado split na função para quebrar a string linha, separando os atributos e facilitando a manipulação.

*4. Lista dos alunos por curso;

Neste caso, é solicitado ao usuário o nome de um curso e o programa imprimi os alunos que estão cadastrados no curso baseado nos atributos dos objetos tipo Professor que foram marcados no arquivo.

*5. Cadastro de Professores;

Nesta opção o usuário digita os dados referentes ao objeto previamente instanciado do tipo Professor (*professor1*, ou somente *professor*) e o programa se encarrega de registrar as entradas em um arquivo *prof.txt* enquanto for solicitado. É análogo ao que foi usado no cadastro dos alunos, só muda o arquivo destino. Exibe.

*6. Remoção de Professores;

Nessa opção, o programa sobrescreve o arquivo de professores que já estava cadastrado, excluindo somente, condicionalmente, a string que foi passada referente ao nome do professor que se quer retirar. Exibe a lista de prof. para conferir.

*7. Listar Professores;

Nessa opção o programa lista os professores cadastrados, separando os seus atributos por virgula: nome, idade, curso que leciona, e um numero de identificação.

*8. Sair;

O programa encerra e deixa uma mensagem na tela, usando template.

2. ESTRUTURA DA SOLUÇÃO

O eixo principal da solução é a classe *BaseDados*. Nela estão os métodos que trabalham as principais funcionalidades do programa e todas a interações com as persistências de dados, sendo o kernel, por assim dizer, do programa. As outras classes ficam basicamente com os métodos padrões getters e setters e os atributos.

É feita uma herança de uma classe base *Pessoa* para as classes *Aluno* e P*rofessor* tornando ela praticamente uma classe abstrata, pois em nenhum momento é instanciado um objeto do tipo Pessoa, servindo somente para fornecer dados, e enxugando muito o código das classes Aluno e Professor que só precisam ter os atributos matrícula/número de identificação, curso e seus getters e setters, respectivamente. Na matrícula e no numero de identificação, são gerados valores aleatórios em uma margem considerável com pouca probabilidade de repetição.

Nos métodos de listagem total, seja de alunos ou professores, é usado uma função split para quebrar a linha dos arquivos onde essa função é parametrizada, ou seja, é usada como base de separação/quebra a *virgula (‘,’)*. Ela vai ate o arquivo e separa os dados separados por virgula em partes para retornar as partes, facilitando a manipulação desses dados. O nome dos alunos e professores aceitam nomes com sobrenomes, pois é usado getline.

Em todos os métodos que abrem arquivos são usados try/catch para informar algum eventual erro que possa ter ocorrido na abertura dos arquivos, deixando mais tranquilo os testes do programa.

Os métodos de interação com o usuário que retornam vectores são 3. Esses métodos são usados para colocar na main esse vetores para que possam ser percorridos por um for e listados da forma que forem solicitados, seja lista completa do conteúdo dos arquivos ou lista parcial, como a listagem dos alunos por curso.

Uma função genérica é usada para retornar valores genéricos na main, e é usada em dois momentos do programa para escrever uma string e integers.

Para usar o programa, entre na pasta do arquivo pelo terminal, digite o comando ‘**make**’ para compilar, depois digite ‘**make clean**’ para limpar os objetos, e por fim ‘**make vai**’ para executar.

3. AS BIBLIOTECAS

BaseDados:

É a classe mais importante do software, onde as coisas funcionam. Primeiramente, ela não tem nenhum atributo, serve como banco de dados e possui os métodos de manipulação desses dados.

Possui um método *Menu*, que simplesmente exibe na tela o menu inicial. Método *cadastrarProfessor* que registra os dados dos professores recebidos, nos arquivos. Método *listarProfessores*, que retorna um vetor de objetos dos professores cadastrados para ser percorrido e exposto para o usuário. Método *removerProfessor* que faz um novo cadastro no arquivo, porém com uma verificação, excluindo o objeto referente à string passada como parâmetro. Método *cadastrarAluno* que escreve no arquivo aluno.txt os dados passados pelo usuário referente aos campos do objeto tipo Aluno. Método *removerAluno*, análogo ao de remover Professor, que sobrescreve uma nova lista no arquivo com a verificação de exclusão da string nome passada. Método *listarAlunos* que lista os Alunos do arquivo na tela, retornando um vector para ser percorrido e imprimindo os campos dos objetos no vetor. Método *listarAlunosCurso* que lista os alunos, chamando o método de listar, porém com uma verificação inclusiva de só povoar o vetor de retorno se o atributo curso do objeto for correspondente à string curso passada. Método split, que retorna um vetor com a string referente aos atributos dos objetos, quebradas quando encontra uma virgula. E, por fim, o método *head* que exibe um cabeçalho para o menu inicial.

Aluno:

É a classe que soma os atributos e métodos herdados de *Pessoa* com um *curso* ao qual o aluno estaria matriculado e um *numero de matricula* da escola. Possui somente atributos e métodos-membros, exceto o “setMatricula()” porque não precisa, no “getMatricula()” ele gera um numero de matricula aleatório.

Professor:

É a classe que soma os atributos e métodos herdados de *Pessoa* com um *curso* ao qual o professor estaria lecionando e um numero *identificador*. Possui somente atributos e métodos-membros, exceto o “setID()” porque não precisa, pois no “getID()” gera um numero identificador aleatório.

Pessoa:

É a classe mais simples, onde só é usada como fonte de dados para as classes *Aluno* e *Professor*. Possui os atributos: *nome* e *idade*, e seus métodos-membros.

4. JUSTIFICATIVA DAS OPÇÕES TOMADAS

As opções foram tomadas com base no melhor encapsulamento do código e da separação dos problemas.

A opção de usar uma classe para armazenar a base de dados foi a melhor para deixar a solução mais dividia, onde o trabalho ‘grosso’ se faz em, basicamente, uma classe, enquanto as outras auxiliam no fornecimento dos dados. Isso ajudou demais e organizou/clareou as ideias, foi a melhor alternativa de todo o trabalho, usar uma classe BaseDados como uma espécie de repositório principal das funções do programa.

A opção de usar vectores de retorno, e não usar arrays em momento algum, veio das dicas do Sidney, que conseguiu passar uma ideia interessante de popular esse vetores para auxiliar no retorno de dados em sequencia, sobretudo na hora de imprimir as listagem dos integrantes da escola. Antes, como relatei na seção das dificuldades, eu pensei em fazer com Arrays, mas os arrays tem tamanho limitado e dificultaria muito, é tanto que eu não tinha achado uma solução decente usando eles, já o vector ele só usa até onde for solicitado no método push_back().

O uso da classe Pessoa como base para Aluno e Professor, veio da necessidade de se usar herança no trabalho, e também enxugou muito o código das classes Aluno e Professor.

O uso do método split, veio da necessidade de se trabalhar com as linhas do arquivo separada pelos atributos do objeto que a linha estava representando.

Com o objetivo de ordenar as listas alfabeticamente, foi avaliado a possibilidade de usar sets ao invés de vectors, mas quando surgiu essa ideia, o trabalho já estava muito avançado e não valeria a pena mudar tudo (justificando um decisão não tomada).

A função template printAlgo, foi usada com retorno genérico, onde printa qualquer tipo de dado que o usuário passar, no exemplo, eu printo string e integers.

5. PRINCIPAIS DIFICULDADES

Primeiramente, tentei fazer o projeto sem muita divisão de tarefas enquanto estudava os conteúdos de POO em C++. Muitos problemas apareceram com as manipulações de arquivos, a interação com o usuário e como guardar as informações (uso de arrays de objetos) etc. Estava complicado modelar uma solução inteligente e que não fugisse do que foi pedido na descrição do trabalho. Pensei, antes, em fazer uma classe *'Classe'* onde nela poderia guardar todas as classes (Aluno, Professor, Curso, etc), mas não tinha muita ideia de como implementaria as soluções de modo que não ficasse muito confuso para trabalhar e para o professor ler.

A questão de como guardar objetos e depois registrá-los nos arquivos também dificultou um pouco, pois, apesar de conhecer o vetor dinâmico(*vector)* , não imaginava como implementar ao meu favor e achei que fosse mais atrapalhar do que ajudar. Pensei no vetor estático mesmo. Foi quando o Sidney Xavier ofereceu ajuda com o trabalho e ensinou algumas técnicas baseada em banco de dados que deixaram as funções muito bem divididas e mostrou como usar o vector o tempo todo como ferramente de intermédio entre o código e os arquivos, o que facilitou muito a vida porque você pode abusar dos vetores que eles sempre são resetados, assim como variáveis auxiliares, e são usados de modo bastante especifico para a tarefa atual. Ele me mostrou alguns códigos base e em cima deles eu fui aprendendo, montando as outras soluções e tentando entender a dinâmica dos vetores com os métodos, os arquivos e as classes. A parte das funções que retornam vector e a função split, que quebra as strings, foram os principais pontos críticos de dificuldade pra mim, mas consegui entender depois de fazer com calma. A primeira vista assusta um pouco, mas depois deu pra perceber que são técnicas muito legais e que depois que você entende a dinâmica tudo fica mais claro.

Foi muito interessantes e esclarecedora a ajuda do Sidney, foi importante nesse trabalho, usarei muitas vezes ainda esses aprendizados em projetos futuros com objetivo de limpar o trabalho e abusar da propriedade de **encapsulamento**.

A disciplina, como um todo, foi difícil pra mim, mas também proporcionou MUITO aprendizado. Quando entrou na parte de orientação a objeto as coisas ficaram diferentes do que eu conhecia e me confundiu um pouco, mas deu pra perceber o poder que esse paradigma tem, principalmente quando vai se trabalhar com maiores volumes de dados, e projetos grandes; envolvendo usuário, arquivos, cadastros etc. Os problemas são diferentes de quando você tem uma questão e escreve uma estrutura linear que a resolve.

REFERÊNCIAS

Sidney Xavier – https://github.com/sidneyxvr

C plus plus - http://www.cplusplus.com/

Youtube – http://youtube.com \ (CFB Cursos, Curso em vídeo e diversos tutoriais).

Google – http://google.com \ diversas buscas, mas principalmente no cplusplus e Stack Overflow.

Slides da disciplina – muitas consultas nos slides.