Instituto Federal de Educação, Ciência e Tecnologia do Norte de Minas Gerais - IFNMG

Curso de Bacharelado em Ciência da Computação

Organização e Sistemas de Arquivos

Prof. Wagner Ferreira de Barros. **Email.:** wagner.barros@ifnmg.edu.br

Prática de Laboratório 2

Organização de Código – Leitura e Escrita de Registros

Descrição:

O objetivo desta prática é ter um primeiro contato com uma proposta de organização ainda simples e ainda comum às práticas de programação vistas no curso de Ciência da Computação, mas com a inclusão de escritas de dados em arquivos de formas talvez não tão usuais.

Resumidamente, os alunos terão que ler um arquivo simples com registros contendo nomes e idades separados por vírgulas e um registro por linha e, ao final, escrever um novo arquivo com registros de tamanho fixo. Deve-se também escrever um método que permita ler arquivos no padrão escrito.

Proposta de organização do código:

Organize seu código em classes, sendo:

- uma classe para representar o registro
- uma classe para representar o arquivo
- uma classe para representar o buffer de entrada e saída

Organize os métodos de cada classe conforme as descrições UML abaixo:

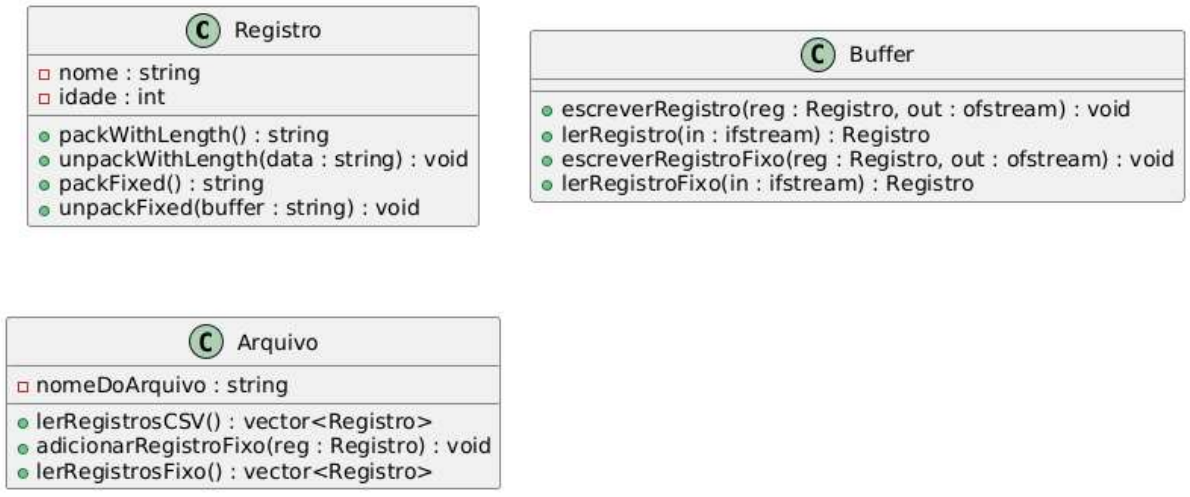

Figura 1: Proposta para as Classes do trabalho incluindo seus atributos e métodos.

A classe *Registro* irá representar o registro e terá métodos para "serializar" e "desserializar" o registro. Este processo irá gerar um serie de bytes correspondente a representação do registro em tamanho fixo, ou seja, irá gerar ou ler uma *string* com o quantidade de bytes máxima da *string* nome e o tamanho de bytes do tipo *int*. Ao inicializar a string a ser serializada (packing), inicialize-a completamente com '\0' e, na hora que for ler de um string serializada (unpacking) para escrever nos atributos da classe, lembre-

se de limpar os '\0' presentes na string (a representação interna do tipo string não usa '\0' para representar o final da cadeia de caracteres, isto é feito apenas em char* ou const char*).

A classe ***Buffer*** será responsável por acessar a *stream* representando o arquivo e ler um conjunto de bytes representando um registro (lerRegistro) ou escrever os bytes de um registro no final da *stream*.

Já a classe ***Arquivo*** irá apenas abrir o arquivo CSV e gerar um vetor de registros (de forma simples, como na aula passada), posteriormente, irá realizar as devidas chamadas as métodos da classe Buffer para realizar a leitura e escrita de registros seguindo os padrões estabelecidos usando os métodos *packing* e *unpacking* da classe Registro.

Os arquivos para teste são os arquivos com extensão CSV vistos na Prática01. Anexos a esta prática.

Observações e dicas de implementação:

- Para melhor eficiência, acostume-se a trabalhar com o arquivo em formato binário sempre que possível.
- Neste início, trabalhe com atributos públicos para evitar a poluição do código com métodos *get* e *set.* Vamos focar em ter um código mais limpo (a segurança na sua utilização vai ficar por conta do programador).
- Utilize funções como a *memcpy* para copiar blocos de dados de qualquer tipo para uma cadeia de caracteres (neste caso, as cadeias de caracteres irão representar os bytes a serem escritos no arquivo, este é um tipo básico na linguagem para fazer tais tipos de operações).
- Ao utilizar as funções *read* e *write* não se esqueça de fazer um *typecasting* para *char** no primeiro argumento com o comando *reinterpret_cast<char*>*, pois este é o tipo de dado padrão a ser lido ou escrito por estas funções.
- Escreva seu código em arquivos separados e utilize um script de makefile para gerenciar a compilação, limpeza e execução do seu projeto.