CENTRO UNIVERSITÁRIO DO RIO SÃO FRANCISCO – UNIRIOS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ALGORITMO E ESTRUTURA DE DADOS

Vanderley José dos Santos

ALGORITMO PARA CADASTRO DE PESSOAS

Paulo Afonso

2024

Vanderley José dos Santos

ALGORITMO PARA CADASTRO DE PESSOAS

Trabalho entregue ao Prof. José Anchieta dos Santos júnior do Curso De Sistema de Informação do Centro Universitário do Rio São Francisco – UNIRIOS.

Paulo Afonso

2024

Sumário

1. Introdução

Algoritmo para cadastro de pessoas com opções para listagem dos usuários cadastrados e/ou exclusão de usuários. Utilizando um arquivo TXT para o armazenamento dos nomes usuários cadastrados.

2. Funcionalidades

2.1. Inclusão das Bibliotecas

“**#include<iostream>**” inclui a biblioteca padrão de entrada e saída do C++ (como cin e cout);

“**#include<fstream>**” esta biblioteca é usada para manipulação de arquivos, como abrir e escrever/ler de arquivos;

“**#include<string>**” permite o uso da classe “**std::string**” para manipular strings de maneira mais prática e segura.

```
#include<iostream> // Para entrada e saída padrão (cin, cout).
#include<fstream> // Para manipulação de arquivos (fstream, ofstream).
#include<string> // Para trabalhar com strings (usando a classe string).
```

“**using namespace std;**” faz com que não seja necessário usar o prefixo “**std::**” ao chamar funções da biblioteca padrão (como “**cin**”, “**cout**”, “**string**”).

```
using namespace std;
```

2.2. Função “**cadastrarUsuario**”:

“**void cadastrarUsuario()**” define uma função que não retorna nenhum valor tipo void.

“**string nome;**” declara uma variável do tipo “**string**” para armazenar o nome do usuário.

“**int idade;**” declara uma variável do tipo inteiro para armazenar a idade do usuário.

Criando Arquivo TXT:

“**ofstream arquivo(“usuarios.txt”, ios::app);**” abre o arquivo o nomeando como “**usuarios**” em modo “**append**” (**ios::app**), ou seja, ele adicionará novos

dados ao final do arquivo sem sobrescrever os existentes. Se o arquivo não existir, ele será criado.

A condição “**if(!arquivo)**” verifica se o arquivo foi aberto corretamente. Se não, exibe uma mensagem de erro e retorna da função, interrompendo o cadastro.

“**cout<<”Digite o nome do usuário: ”;**” exibe uma mensagem solicitando o nome do usuário.

“**getline(cin, nome);**” lê uma linha inteira do teclado (permitindo espaços) e armazena na variável “**nome**”.

“**cout<<”Digite a idade do usuário: ”;**” exibe uma mensagem pedindo a idade do usuário.

“**cin>>idade;**” lê um número inteiro e o armazena em “**idade**”.

“**cin.ignore()**” limpa o buffer de entrada, o que é necessário para evitar problemas ao usar “**getline()**” depois, pois o “**cin**” pode deixar uma caractere de nova linha no buffer que poderia ser lido como entrada.

“**arquivo<<nome<<”,”<<idade<<endl;**” escreve o nome, seguido de uma vírgula, e a idade no arquivo “**usuario.txt**”. O “**endl**” é usado para adicionar uma nova linha após os dados.

“**cout**” exibe uma mensagem de sucesso.

“**arquivo.close();**” fecha o arquivo depois de escrever os dados.

```
void cadastrarUsuario(){
    string nome;
    int idade;
    // Abrindo o arquivo em modo append para adicionar novos usuários
    ofstream arquivo("usuarios.txt", ios::app);
    if(!arquivo){
        cout<<"Erro ao abrir o arquivo!"<<endl;
        return;
    }
    cout<<"Digite o nome do usuário: ";
    getline(cin, nome);
    cout<<"Digite a idade do usuário: ";
    cin>>idade;
    cin.ignore(); // Limpar o buffer do cin.
    // Escrever os dados no arquivo.
    arquivo<<nome<<","<<idade<<endl;
    cout<<"Usuário cadastrado com sucesso!"<<endl;
    arquivo.close();
}
```

2.3. Função “listarUsuarios()”:

“**ifstream arquivo(“usuários.txt”);**” abre o arquivo “**usuários.txt**” para leitura.

Se o arquivo não for aberto corretamente, a condição “**if(!arquivo)**” exibe uma mensagem de erro e a função retorna, interrompendo a listagem dos usuários.

“**cout<<”\nLista de usuários cadastrados:\n”;**” exibe uma mensagem que indica que a lista de usuários será mostrada.

O laço “**while(getline(arquivo, nome, ‘,’)&& arquivo>>idade)**” lê do arquivo:

“**...getline(arquivo, nome, ‘,’)...**” lê até encontrar uma vírgula, que separa o nome da idade. O nome é armazenado na variável nome.

“**...arquivo>>idade...**” lê o valor da idade (após a vírgula) e o armazena na variável “**idade**”.

A função “**arquivo.ignore()**” é chamada para garantir que o caractere de nova linha seja ignorado após a leitura da idade, garantindo que a próxima iteração do laço funcione corretamente.

Após cada iteração, os dados de nome e idade são exibidos no console. Ao final do laço, “**arquivo.close()**” fecha o arquivo.

```
24  void listarUsuarios(){
25      ifstream arquivo("usuarios.txt");
26      string nome;
27      int idade;
28      if(!arquivo){
29          cout<<"Erro ao abrir o arquivo!"<<endl;
30          return;
31      }
32      cout<<"\nLista de usuários cadastrados:\n";
33      while(getline(arquivo, nome, ',')&&arquivo>>idade){
34          arquivo.ignore(); // Ignorar a quebra de linha.
35          cout<<"Nome: "<<nome<<", Idade: "<<idade<<endl;
36      }
37      arquivo.close();
38  }
```

2.4. Função “excluirUsuario()”:

A função “**excluirUsuario()**” é responsável por excluir um usuário.

A variável “**nomeExcluir**” armazenará o nome do usuário que será excluído.

“**getline(cin, nomeExcluir)**” lê o nome do usuário a ser excluído.

“**ifstream arquivo(“usuários.txt”);**” abre o arquivo “**usuários.txt**” para leitura e “**ofstream temp(“temp.txt”);**” cria um arquivo temporário “**temp.txt**” para armazenar os dados atualizados (sem o usuário a ser excluído).

A condição “**if(!arquivo || !temp)**” será verdadeira se qualquer um dos arquivos não for aberto corretamente, exibe uma mensagem de erro e retorna da função.

“**string nome;**” armazena o nome lido do arquivo.

“**int idade;**” armazena a idade lida do arquivo.

“**bool encontrado=false;**” verifica se o usuário a ser excluído foi encontrado.

O laço “**while(getline(arquivo, nome, ‘,’) && arquivo>>idade)**” lê o arquivo da mesma forma que na função “**listarUsuario()**”.

Se o nome lido não for igual ao nome a ser excluído “(nome!=nomeExcluir)”, os dados são copiados para o arquivo “temp”. Caso contrário, “encontrado” é setado para “true”, indicando que o usuário foi encontrado.

Após o laço, os arquivos são fechados: “**arquivo.close()**”, “**temp.close()**”.

Se o usuário foi encontrado, o arquivo original “**usuários.txt**” é removido e o arquivo temporário “**temp.txt**” é renomeado para “**usuários.txt**”. Se o usuário não foi encontrado, exibe uma mensagem de erro e remove o arquivo temporário.

```
void excluirUsuario(){
    string nomeExcluir;
    cout<<"Digite o nome do usuário a ser excluído: ";
    getline(cin, nomeExcluir);
    ifstream arquivo("usuarios.txt");
    ofstream temp("temp.txt"); // Arquivo temporário para reescrever os dados.
    if(!arquivo||!temp){
        cout<<"Erro ao abrir o arquivo!"<<endl;
        return;
    }
    string nome;
    int idade;
    bool encontrado = false;
    // Ler o arquivo original e copiar para o arquivo temporário.
    while(getline(arquivo, nome, ',')&&arquivo>>idade){
        arquivo.ignore(); // Ignorar a quebra de linha.
        if(nome!=nomeExcluir){
            // Se o nome não for o que queremos excluir, copiar para o arquivo temporário.
            temp<<nome<<","<<idade<<endl;
        }else{
            encontrado=true; // Encontramos o usuário a ser excluído.
        }
    }
    arquivo.close();
    temp.close();
    // Se o usuário foi encontrado, substituímos o arquivo original pelo temporário.
    if(encontrado){
        remove("usuarios.txt"); // Apagar o arquivo original.
        rename("temp.txt", "usuarios.txt"); // Renomear o arquivo temporário para o nome do arquivo original.
        cout<<"Usuário excluído com sucesso!"<<endl;
    }else{
        cout<<"Usuário não encontrado!"<<endl;
        remove("temp.txt"); // Se não encontrou, apaga o arquivo temporário.
    }
}
```

2.5. Função main()

“**int escolha;**” declara uma variável para armazenar a opção escolhida pelo usuário no menu.

“**do{...}while(...)**” exibe um menu de opções até que o usuário escolha a opção ‘**Sair**’ (opção 4).

“**cin>>escolha;**” lê a escolha do usuário.

“**cin.ignore();**” limpa o buffer de entrada.

“**switch(escolha){...}**” decide qual função chamar com base na opção escolhida:

“**case 1**” chama a função “**cadastrarUsuario()**”.

“**case 2**” chama a função “**listarUsuarios()**”.

“**case 3**” chama a função “**excluirUsuario()**”.

“**case 4**” exibe uma mensagem dizendo que o programa está saindo.

“**default:**” exibe uma mensagem de erro se a opção for inválida.

“**return 0;**” retorna 0, indicando que o programa terminou com sucesso.

```
74  int main(){
75      int escolha;
76      do{
77          cout<<"\nMenu de opções:\n";
78          cout<<"1. Cadastrar usuário\n";
79          cout<<"2. Listar usuários\n";
80          cout<<"3. Excluir usuário\n";
81          cout<<"4. Sair\n";
82          cout<<"Escolha uma opção: ";
83          cin>>escolha;
84          cin.ignore(); // Limpar o buffer do cin.
85          switch(escolha){
86              case 1:
87                  cadastrarUsuario();
88                  break;
89              case 2:
90                  listarUsuarios();
91                  break;
92              case 3:
93                  excluirUsuario();
94                  break;
95              case 4:
96                  cout<<"Saindo...\n";
97                  break;
98              default:
99                  cout<<"Opção inválida!\n";
100         }
101     }while(escolha!=4);
102     return 0;
103 }
```

3. Conclusão

Este código oferece um pequeno sistema de gerenciamento de usuários usando arquivos de texto, com funções para cadastrar, listar e excluir usuários. Ele utiliza manipulação de arquivos, entrada/saída de dados e controle de fluxo básico com “**switch**” e “**while**”.

Sobre possíveis melhorias, há a possibilidade de melhora na validação de entradas, na leitura e escrita de arquivos mais eficientemente, no uso de IDs únicos para usuários para evitar problemas com nomes duplicados, exibição de mensagens de erro mais detalhadas, melhorias na interface de usuário (confirmar exclusões e permitir edição de dados).

Essas melhorias podem não ser necessárias em um projeto pequeno, mas são boas práticas caso o sistema se expanda ou se torna mais complexo.