Universidade Zambeze

Faculdade de Ciências e Tecnologias

Disciplina: Informática II

1º ano 2º semestre – Laboral

Relatório de Desenvolvimento e Funcionamento do Sistema de Controle de Estoque

Aplicação de Cpp no Monitoramento

Autores

Baptista Elidio Fernando Jaime

Humberto Francisco Rodrigues

Marcelo Casquinha

Docente:

Francisco Chimbinde

11 de novembro de 2024

Sumário

Sumário de ilustrações

Introdução

Este projeto consiste no desenvolvimento de um sistema de controle de estoque em C++. O sistema permite realizar operações comuns de gestão de estoque, como cadastrar produtos, listar produtos cadastrados, atualizar e excluir informações, além de salvar e carregar os dados de estoque a partir de um arquivo binário. Para a implementação, utilizamos conceitos de orientação a objetos, como encapsulamento e modularização, e fizemos uso de estruturas de dados dinâmicas e manipulação de arquivos.

Objetivo

O objetivo deste projeto é fornecer uma solução prática e eficiente para o gerenciamento de produtos em estoque. O sistema pode ser utilizado em uma pequena empresa para facilitar o monitoramento de produtos, evitando perdas, excesso ou falta de produtos no estoque. Além disso, o projeto visa consolidar o aprendizado de programação em C++, especialmente em relação ao uso de classes, vetores dinâmicos, ponteiros e arquivos.

Estrutura do Código

O sistema foi desenvolvido em duas classes principais:

1. **Classe `Produto`:**
    - A classe `Produto` representa o produto armazenado em estoque, contendo os atributos `id`, `nome`, `preco` e quantidade.
    - Os atributos são usados para identificar e descrever o produto, o que facilita as operações de manipulação, como atualização e exclusão.

```
6  class Produto {
7  public:
8      int id;
9      std::string nome;
10     float preco;
11     int quantidade;
12
13     Produto(int id, const std::string& nome, float preco, int quantidade)
14         : id(id), nome(nome), preco(preco), quantidade(quantidade) {}
15 };
16
```

Figura 1 classe Produto

2. **Classe `ControleEstoque`:**
    - A classe ControleEstoque encapsula todas as operações de gerenciamento de produtos, incluindo cadastro, listagem, atualização, exclusão e operações de entrada e saída de dados em arquivos.
    - Um vetor dinâmico (std::vector) armazena os objetos Produto, o que facilita o redimensionamento automático do vetor conforme novos produtos são cadastrados.
    - A variável proximo_id é utilizada para controlar e atribuir IDs exclusivos para cada produto novo.

```
17 class ControleEstoque {
18 private:
19     std::vector<Produto> estoque;
20     int proximo_id = 1;
21
```

Figura 2 classe Control de Estoque

Funcionalidades Implementadas

1. **Cadastro de Produto**
    - A função cadastrarProduto() solicita o nome, preço e quantidade do produto, cria um objeto Produto com essas informações e adiciona-o ao vetor estoque.

- A cada novo produto, o id é automaticamente incrementado, garantindo unicidade para cada produto cadastrado.

```
public:
    void cadastrarProduto() {
        std::string nome;
        float preco;
        int quantidade;

        std::cout << "Nome do produto: ";
        std::getline(std::cin >> std::ws, nome);
        std::cout << "Preço do produto: ";
        std::cin >> preco;
        std::cout << "Quantidade do produto: ";
        std::cin >> quantidade;

        Produto novo_produto(proximo_id++, nome, preco, quantidade);
        estoque.push_back(novo_produto);

        std::cout << "Produto cadastrado com sucesso! ID do produto: " << novo_produto.id << "\n";
    }
```

3 Funcionalidade cadastro

2. Listagem de Produtos

- A função listarProdutos() percorre o vetor estoque e exibe os dados de cada produto.
- Caso o estoque esteja vazio, o sistema informa ao usuário que não há produtos cadastrados.

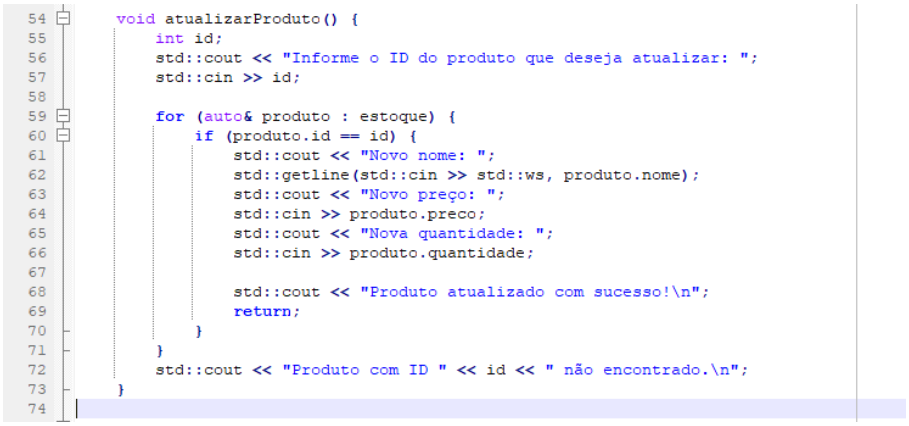
```
    void listarProdutos() const {
        if (estoque.empty()) {
            std::cout << "Nenhum produto cadastrado.\n";
            return;
        }

        std::cout << "Lista de Produtos:\n";
        for (const auto& produto : estoque) {
            std::cout << "ID: " << produto.id << ", Nome: " << produto.nome
                      << ", Preço: " << produto.preco << ", Quantidade: " << produto.quantidade << "\n";
        }
    }
```

4 Funcionalidade listar produto

3. Atualização de Produto

- A função atualizarProduto() solicita um id do produto a ser atualizado. Caso o produto exista, o usuário pode alterar seu nome, preço e quantidade.
- Esse método permite a atualização dos dados sem alterar a ordem ou o ID do produto, mantendo a integridade dos dados no sistema.

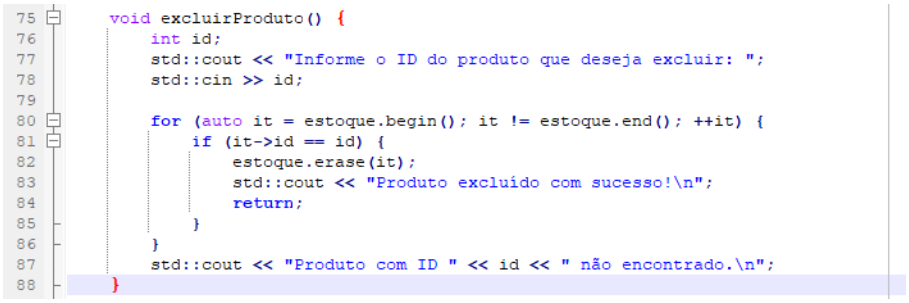


```
54    void atualizarProduto() {
55        int id;
56        std::cout << "Informe o ID do produto que deseja atualizar: ";
57        std::cin >> id;
58
59        for (auto& produto : estoque) {
60            if (produto.id == id) {
61                std::cout << "Novo nome: ";
62                std::getline(std::cin >> std::ws, produto.nome);
63                std::cout << "Novo preço: ";
64                std::cin >> produto.preco;
65                std::cout << "Nova quantidade: ";
66                std::cin >> produto.quantidade;
67
68                std::cout << "Produto atualizado com sucesso!\n";
69                return;
70            }
71        }
72        std::cout << "Produto com ID " << id << " não encontrado.\n";
73    }
74
```

5 Funcionalidade atualização de produtos

4. Exclusão de Produto

- A função excluirProduto() remove um produto do vetor com base no id fornecido pelo usuário.
- A exclusão é realizada deslocando os produtos subsequentes no vetor para "cobrir" o produto excluído, mantendo a estrutura de armazenamento sem posições vazias.



```
75    void excluirProduto() {
76        int id;
77        std::cout << "Informe o ID do produto que deseja excluir: ";
78        std::cin >> id;
79
80        for (auto it = estoque.begin(); it != estoque.end(); ++it) {
81            if (it->id == id) {
82                estoque.erase(it);
83                std::cout << "Produto excluído com sucesso!\n";
84                return;
85            }
86        }
87        std::cout << "Produto com ID " << id << " não encontrado.\n";
88    }
```

6 Funcionalidade exclusão de produtos

5. Salvamento e Carregamento de Dados

- A função salvarEmArquivo() salva o vetor estoque em um arquivo binário (estoque.dat). Para cada produto, são salvos o id, nome, preco e quantidade.
- A função carregarDoArquivo() carrega os dados do arquivo binário para o vetor estoque ao iniciar o sistema. Se o arquivo não for encontrado, o sistema inicia com um estoque vazio.
- Essa persistência de dados permite que as informações sejam preservadas entre execuções, garantindo que o sistema possa ser fechado e reaberto sem perda de dados.

6. **Menu Interativo**
    - A função menu() exibe um menu com as opções de cada funcionalidade e controla o fluxo do programa de acordo com a escolha do usuário.
    - O menu permite que o usuário navegue por cada funcionalidade de maneira intuitiva.

```
142     void menu() {
143         int opcao;
144         do {
145             std::cout << "\nControle de Estoque\n";
146             std::cout << "1. Cadastrar Produto\n";
147             std::cout << "2. Listar Produtos\n";
148             std::cout << "3. Atualizar Produto\n";
149             std::cout << "4. Excluir Produto\n";
150             std::cout << "5. Salvar Dados\n";
151             std::cout << "6. Carregar Dados\n";
152             std::cout << "0. Sair\n";
153             std::cout << "Escolha uma opção: ";
154             std::cin >> opcao;
155
```

7 *Funcionalidade menu interativo*

```
156             switch (opcao) {
157                 case 1:
158                     cadastrarProduto();
159                     break;
160                 case 2:
161                     listarProdutos();
162                     break;
163                 case 3:
164                     atualizarProduto();
165                     break;
166                 case 4:
167                     excluirProduto();
168                     break;
169                 case 5:
170                     salvarEmArquivo();
171                     break;
172                 case 6:
173                     carregarDoArquivo();
174                     break;
175                 case 0:
176                     std::cout << "Saindo do sistema.\n";
177                     break;
178                 default:
179                     std::cout << "Opção inválida! Tente novamente.\n";
180             }
181         } while (opcao != 0);
```

8 *Funcionalidade menu interativo*

Exemplos de Uso

Ao iniciar o programa, o usuário tem acesso ao menu interativo. Abaixo estão exemplos de uso das funcionalidades:

1. **Cadastro**: O usuário escolhe a opção 1, insere o nome, preço e quantidade. O sistema exibe o ID do produto cadastrado.
2. **Listagem**: Escolhendo a opção 2, o sistema exibe todos os produtos em estoque, com ID, nome, preço e quantidade.

3. **Atualização**: Na opção 3, o usuário insere o ID do produto que deseja atualizar e insere os novos valores.
4. **Exclusão**: Na opção 4, o usuário insere o ID do produto que deseja excluir, e o sistema confirma a exclusão.
5. **Salvar Dados**: A opção 5 salva os dados no arquivo binário estoque.dat.
6. **Carregar Dados**: A opção 6 carrega os dados do arquivo, permitindo a continuação de um estoque previamente salvo.

Sistema de Controle de Estoque Código Fonte:

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string>

class Produto {
public:
    int id;
    std::string nome;
    float preco;
    int quantidade;

    Produto(int id, const std::string& nome, float preco, int quantidade)
        : id(id), nome(nome), preco(preco), quantidade(quantidade) {}
};

class ControleEstoque {
private:
    std::vector<Produto> estoque;
    int proximo_id = 1;

public:
    void cadastrarProduto() {
        std::string nome;
        float preco;
        int quantidade;
```

```
        std::cout << "Nome do produto: ";
        std::getline(std::cin >> std::ws, nome);
        std::cout << "Preço do produto: ";
        std::cin >> preco;
        std::cout << "Quantidade do produto: ";
        std::cin >> quantidade;

        Produto novo_produto(proximo_id++, nome, preco, quantidade);
        estoque.push_back(novo_produto);

        std::cout << "Produto cadastrado com sucesso! ID do produto: " <<
novo_produto.id << "\n";
    }

    void listarProdutos() const {
        if (estoque.empty()) {
            std::cout << "Nenhum produto cadastrado.\n";
            return;
        }

        std::cout << "Lista de Produtos:\n";
        for (const auto& produto : estoque) {
            std::cout << "ID: " << produto.id << ", Nome: " << produto.nome
                      << ", Preço: " << produto.preco << ", Quantidade: " <<
produto.quantidade << "\n";
        }
    }

    void atualizarProduto() {
        int id;
        std::cout << "Informe o ID do produto que deseja atualizar: ";
        std::cin >> id;

        for (auto& produto : estoque) {
            if (produto.id == id) {
                std::cout << "Novo nome: ";
```

```
            std::getline(std::cin >> std::ws, produto.nome);
            std::cout << "Novo preço: ";
            std::cin >> produto.preco;
            std::cout << "Nova quantidade: ";
            std::cin >> produto.quantidade;

            std::cout << "Produto atualizado com sucesso!\n";
            return;
        }
    }
    std::cout << "Produto com ID " << id << " não encontrado.\n";
}

void excluirProduto() {
    int id;
    std::cout << "Informe o ID do produto que deseja excluir: ";
    std::cin >> id;

    for (auto it = estoque.begin(); it != estoque.end(); ++it) {
        if (it->id == id) {
            estoque.erase(it);
            std::cout << "Produto excluído com sucesso!\n";
            return;
        }
    }
    std::cout << "Produto com ID " << id << " não encontrado.\n";
}

void salvarEmArquivo() const {
    std::ofstream arquivo("estoque.dat", std::ios::binary);
    if (!arquivo) {
        std::cout << "Erro ao abrir o arquivo.\n";
        return;
    }
```

```
        int total_produtos = estoque.size();
        arquivo.write(reinterpret_cast<const char*>(&total_produtos),
sizeof(total_produtos));

        for (const auto& produto : estoque) {
            arquivo.write(reinterpret_cast<const char*>(&produto.id),
sizeof(produto.id));
            int nome_size = produto.nome.size();
            arquivo.write(reinterpret_cast<const char*>(&nome_size),
sizeof(nome_size));
            arquivo.write(produto.nome.c_str(), nome_size);
            arquivo.write(reinterpret_cast<const char*>(&produto.preco),
sizeof(produto.preco));
            arquivo.write(reinterpret_cast<const char*>(&produto.quantidade),
sizeof(produto.quantidade));
        }

        std::cout << "Dados salvos com sucesso!\n";
    }

    void carregarDoArquivo() {
        std::ifstream arquivo("estoque.dat", std::ios::binary);
        if (!arquivo) {
            std::cout << "Nenhum arquivo de dados encontrado. Iniciando
estoque vazio.\n";
            return;
        }
        int total_produtos;
        arquivo.read(reinterpret_cast<char*>(&total_produtos),
sizeof(total_produtos));

        estoque.clear();
        for (int i = 0; i < total_produtos; ++i) {
            int id, quantidade, nome_size;
            float preco;
            std::string nome;

            arquivo.read(reinterpret_cast<char*>(&id), sizeof(id));
```

```
            arquivo.read(reinterpret_cast<char*>(&nome_size),
sizeof(nome_size));
            nome.resize(nome_size);
            arquivo.read(&nome[0], nome_size);
            arquivo.read(reinterpret_cast<char*>(&preco), sizeof(preco));
            arquivo.read(reinterpret_cast<char*>(&quantidade),
sizeof(quantidade));

            estoque.emplace_back(id, nome, preco, quantidade);
            if (id >= proximo_id) proximo_id = id + 1;
        }

        std::cout << "Dados carregados com sucesso!\n";
    }
    void menu() {
        int opcao;
        do {
            std::cout << "\nControle de Estoque\n";
            std::cout << "1. Cadastrar Produto\n";
            std::cout << "2. Listar Produtos\n";
            std::cout << "3. Atualizar Produto\n";
            std::cout << "4. Excluir Produto\n";
            std::cout << "5. Salvar Dados\n";
            std::cout << "6. Carregar Dados\n";
            std::cout << "0. Sair\n";
            std::cout << "Escolha uma opção: ";
            std::cin >> opcao;

            switch (opcao) {
                case 1:
                    cadastrarProduto();
                    break;
                case 2:
                    listarProdutos();
                    break;
                case 3:
```

```
                    atualizarProduto();
                    break;
                case 4:
                    excluirProduto();
                    break;
                case 5:
                    salvarEmArquivo();
                    break;
                case 6:
                    carregarDoArquivo();
                    break;
                case 0:
                    std::cout << "Saindo do sistema.\n";
                    break;
                default:
                    std::cout << "Opção inválida! Tente novamente.\n";
            }
        } while (opcao != 0);
    }
};
int main() {
    ControleEstoque controleEstoque;
    controleEstoque.carregarDoArquivo();  // Carregar dados ao iniciar o
sistema
    controleEstoque.menu();               // Executar o menu interativo
    controleEstoque.salvarEmArquivo();    // Salvar dados ao sair do sistema
    return 0;
}
```

Conclusão

O sistema de controle de estoque desenvolvido apresenta todas as funcionalidades necessárias para gerenciar produtos de forma eficiente e prática. A utilização de C++ possibilitou a implementação de uma estrutura modular, com classes bem definidas e persistência de dados em arquivos binários.

Como possível melhoria, o sistema poderia incluir uma interface gráfica, o que tornaria o uso mais amigável para o usuário final. Além disso, seria interessante adicionar relatórios mais detalhados e filtros de busca, permitindo a consulta por critérios como intervalo de preços ou quantidade mínima.

O projeto cumpriu seus objetivos e proporcionou um aprofundamento prático nas técnicas de manipulação de dados e arquivos, além de conceitos avançados de C++.