UFOP – UNIVERSIDADE FEDERAL DE OURO PRETO

DECOM – DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

BCC221 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Trabalho Prático 1:
C++

BRUNA CRISTINA GONCALVES - 20.2.4123

ELISA ALVES VELOSO - 20.2.4006

LAVÍNIA FONSECA PEREIRA - 19.1.4170

LUCAS GOMES DOS SANTOS - 20.1.4108

Ouro Preto – MG
2023

SUMÁRIO

1. INTRODUÇÃO

A Programação Orientada a Objetos tem como objetivo aproximar a lógica da programação aos acontecimentos mundanos, uma vez que os objetos e classes criados a partir dos modelos representam objetos da realidade que processam os dados. Assim, classes e objetos encapsulam dados e métodos de modo a simplificar programas, organizar o código e facilitar seu entendimento.

O presente trabalho tem o intuito de implementar um sistema que permita realizar o cadastro de funcionários e realizar o controle de ponto dos mesmos.

Na implementação, faz-se presente o uso de sobrecarga de operadores, e o sistema foi codificado no main, onde as operações descritas e implementadas são chamadas, possibilitando a manipulação e transformação para futuras operações.

2. OBJETIVO

O objetivo deste trabalho prático consiste em implementar um programa em C++ seguindo as instruções necessárias para desenvolver um sistema de cadastro de funcionários e realizar o controle de ponto destes funcionários.

3. IMPLEMENTAÇÃO

Na implementação, foram utilizados componentes da STL (Standard Template Library). Ela é um conjunto de classes de template C++ para fornecer estruturas de dados de programação comuns e funções, como listas, pilhas, arrays, etc. É uma biblioteca generalizada e, portanto, seus componentes são parametrizados. Ela conta com os chamados containers, que são estruturas conhecidas, implementadas e testadas que evitam a repetição de código e reduz a possibilidade de erros.

O container vector foi o principal utilizado na criação das classes, sendo mais específica na criação de um vetor para armazenar os funcionários cadastrados, já que, além de ser o mais familiar ao grupo, é eficiente para se acessar elementos específicos nele contidos. O vector é uma estrutura que funciona semelhante ao vetor, com métodos já implementados, de modo a facilitar seu uso.

Outra vantagem do vector é o modo de inserção. Nessa estrutura, a inserção no final tem custo O(1), de modo que seu uso é mais proveitoso.

Para a realização dos testes, fomos testando parte a parte, visto que, foi necessário criar várias telas para o sistema, dessa forma, quando o programa foi criado por completo, não foi necessário criar um txt,

Para a implementação, segue em cada subseção as explicações de cada arquivo presente no código fonte.

3.1 CHEFE

O construtor da classe *Chefe*, recebe três strings (*nome, usuário e senha*) e chama o construtor da classe base *Pessoa* para inicializar os membros herdados.

3.2 FUNCIONÁRIO

Começa a definição do construtor da classe Funcionario. Ele recebe vários parâmetros, incluindo o nome, usuário, senha, função do funcionário, salário por hora, horas trabalhadas, horas extras e horas pendentes. Além disso, ele chama o construtor da classe base Pessoa para inicializar os membros herdados. Os valores iniciais de horasTrabalhadas, horasExtra e horasPendentes são definidos como 0.0, 0.0 e 40.0, respectivamente.

Em seguida teremos os getters e setters. Os getters retornam a função como uma string, e o const no final da função indica que ela não modificará os membros da classe. Os setters, define uma função membro, que atualiza a função com base em uma string fornecida como argumento.

Na função cadastraPonto, permite que um funcionário registre seu ponto de entrada e saída. Ela solicita ao usuário que digite o horário de início e o horário final do expediente, em formato HH:MM. Em seguida, ela realiza cálculos para determinar a duração do expediente, verifica se houve horas extras, atualiza os membros horasTrabalhadas, horasExtra e horasPendentes com base nos cálculos e exibe uma mensagem indicando que o ponto foi registrado com sucesso.

3.3 MAIN

Em resumo, no main é onde colocamos todo o menu. Primeiramente, foi criado um objeto MenuChefe para realizar operações relacionadas ao menu do chefe. Se o usuário e a senha forem válidos, um novo loop while é iniciado para exibir o menu do chefe. O menu do chefe é exibido com opções como cadastrar funcionário, listar funcionários, checar ponto, cálculo do salário, etc. O usuário escolhe uma opção digitando um número. Dependendo da opção escolhida, diferentes ações são executadas. Por exemplo, se o usuário escolher cadastrar funcionário, é chamado o método cadastrarFuncionario do objeto menuChefe para criar um novo funcionário. O funcionário criado é adicionado ao vetor funcionarios e uma mensagem é exibida informando o sucesso ou falha do cadastro. O menu do funcionário é exibido com opções como cadastrar ponto, exibir salário detalhado, cadastrar venda, listar vendas, etc. O usuário escolhe uma opção digitando um número. Dependendo da opção escolhida, diferentes ações são executadas. Por exemplo, se o usuário escolher cadastrar ponto, é chamado o método cadastraPonto do funcionário autenticado

3.4 MENUCHEFE

Na função MenuChefe::cadastrarFuncionario, são solicitados dados como nome, usuário, senha, função, salário por hora, entre outros, para cadastrar um novo funcionário. Com base na função informada, é criado um objeto Vendedor ou Supervisor utilizando o operador new. Em seguida, o ponteiro para o objeto do funcionário criado é retornado. A função MenuChefe::listarFuncionarios recebe um vetor de ponteiros para Funcionario e percorre cada elemento, imprimindo informações como nome, função e salário por hora de cada funcionário. A função MenuChefe::checarPonto recebe uma referência para um objeto Funcionario e exibe informações sobre o ponto do funcionário, como nome, horas trabalhadas, horas extras e horas pendentes.

3.5 PESSOA

Essa classe Pessoa serve como uma classe base que pode ser estendida por outras classes para representar diferentes tipos de pessoas, como funcionários, clientes, administradores, etc. Ela fornece métodos para acessar e

modificar as informações básicas de uma pessoa, além de um método para verificar a autenticação com base no usuário e senha fornecidos.

3.6 SUPERVISOR

A classe Supervisor, é derivada da classe base ‘Funcionário’ e ela adiciona funcionalidades específicas para um supervisor de vendas, como o gerenciamento dos vendedores sob sua responsabilidade, o cálculo do total de vendas dos vendedores e o cálculo do salário total do supervisor considerando suas horas trabalhadas e as bonificações dos vendedores.

3.7 VENDEDOR

A classe Vendedor, é derivada da classe base ‘Funcionário’ ela adiciona funcionalidades específicas para um vendedor, como a adição de vendas, o cálculo do salário total considerando as horas trabalhadas e as vendas realizadas, e o acesso ao valor das vendas.

4. ANÁLISES

4.1 IDE UTILIZADA

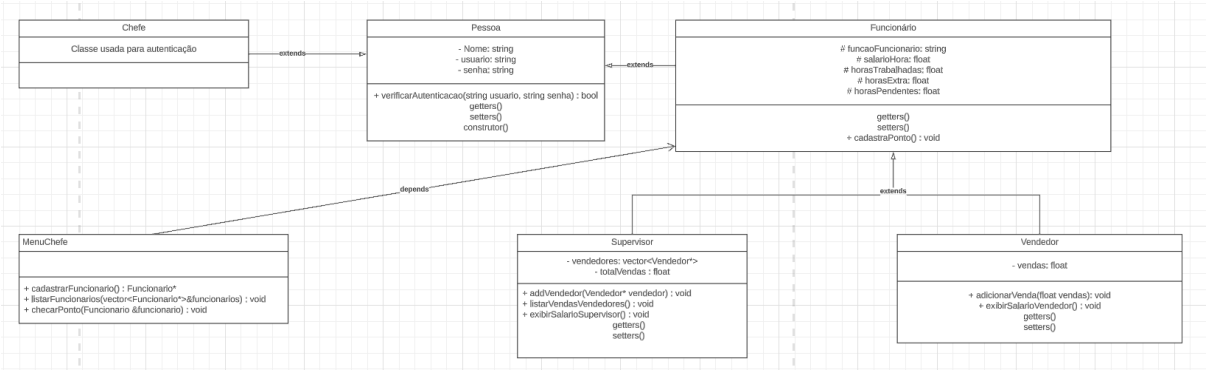
Considerando que existem diversos ambientes para execução de programa, e um programa pode executar em um ambiente e não executar em outro, é necessário deixar explícito no relatório qual IDE utilizamos: Visual Studio Code.

4.2 COMO OPERAR O SISTEMA

Foi criado um arquivo "compilar.sh", basta rodar ./compilar.sh e criará um executável, depois basta rodar “ ./executavel ” para iniciar o código. (usando linux)

Para compilar na mão, basta seguir a seguinte instrução: g++ Chefe.cpp Funcionario.cpp main.cpp MenuChefe.cpp Pessoa.cpp Supervisor.cpp Vendedor.cpp -o executavel

4.3 UNIFIED MODELING LANGUAGE (UML)



[Acesso ao UML]

4.4 CONCEITOS UTILIZADOS

Foi utilizado os conceitos aprendidos em sala no geral, tais como, classes, getters, setters, sobrecarga, vector, ponteiro, entre outros.

4.5 ESTRATÉGIAS UTILIZADAS PARA O TESTE

Foi utilizado testes de fluxos com vendedores e supervisores, inserindo os requisitos necessários para realizar testes e comparar ao final se todas restrições estavam bem estabelecidas.

4.6 ERROS OCORRIDOS

Houve apenas erros básicos de implementação (lógica), nada que o material da disciplina não ajudasse a solucionar tais problemas.

4.7 DIFICULDADES

A dificuldade maior foi com a modelagem das classes para implementação de herança, e para desenvolver as restrições.

4.8 O QUE NÃO FOI IMPLEMENTADO

No presente trabalho foi implementado todos os requisitos listados no PDF de orientação.

4.9 IMPLEMENTAÇÃO EXTRA

Como não foi solicitado a opção de adicionar um vendedor a um supervisor, resolvemos adicionar essa opção no nosso menu de funcionários com intuito de ser mais prático e fácil para saber quantos vendedores um determinado supervisor terá, para que no final da exibição da lista final com os dados de um determinado funcionário listar também qual supervisor está relacionado a ele. Fizemos também um makefile para rodar mais fácil o código.

5. ANÁLISE GERAL

O desenvolvimento dos códigos presentes nesse trabalho possibilitaram o maior conhecimento prático acerca das técnicas e lógicas da programação na linguagem C++. Tendo isso em vista, a linguagem em questão se aproxima bastante da lógica computacional em si, ou seja, possui um médio nível de abstração. Nesse sentido, o nível nesse contexto é entendido como o detalhamento do algoritmo para a execução das tarefas. Tendo em vista o detalhamento em C++, é perceptível que esse deve ser mais detalhado para que o programa execute da maneira que os desenvolvedores querem, e para o aprendizado da programação, faz sentido por estarmos mais próximos do que a máquina entende ao ler cada linha de código.

6. IMPRESSÕES GERAIS

O trabalho prático I apresentou a necessidade de que sejam implementados os conhecimentos teóricos aprendidos em sala. No final, sentimos que entendemos mais certamente acerca da implementação do que foi passado em sala de aula, sendo que os principais aprendizados concretizados foram sobre classes e seus relacionamentos, construtores, destrutores, headers, composição, sobrecarga de operadores, herança, polimorfismo..., no qual vimos os conceitos sobre cada qual em sala de aula.

7. CONSIDERAÇÕES FINAIS

A partir da implementação e testagem das propostas desse trabalho, pode-se concluir que os recursos da STL são de extrema utilidade no paradigma da

orientação a objetos, tendo suas vantagens de evitar erros, redundância de código e facilitar a manipulação de dados. Com o uso de containers, consegue-se armazenar e trabalhar sobre as informações de modo facilitado. Por fim, os conceitos aprendidos teoricamente em sala de aula acerca do uso das estruturas de programação a objetos na linguagem C++ foram aplicados e aprendidos a partir do exercício prático do uso de tais.

8. CONCLUSÃO

Dessa forma, podemos concluir diversas coisas sobre o paradigma da orientação a objetos com esse trabalho. Nele vimos sobre diversos aspectos da STL, e sua utilidade para evitar erros, repetição de código e facilitação de manipulação de dados. Com os containers conseguimos armazenar e trabalhar sobre nossas informações de maneira muito fácil, utilizando os iteradores para nos movermos sobre ele.

O desenvolvimento das análises feitas no código, bem como as diferentes verificações para as soluções das operações, foram importantes para a concretização do aprendizado sobre programação orientada a objetos. Ao apresentar o desafio de fazer verificações e propor soluções. Além disso, o trabalho nos colocou numa posição de sair do pensamento mais específico e pensar de forma ampla para que todas as regras sejam atendidas e o programa respondesse corretamente às linhas de código.

Por fim, o presente trabalho foi muito proveitoso a todo grupo, pois nos permitiu exercitar um pouco alguns dos conceitos vistos na aula(classes, herança, sobrecargas, containers, iterators, algoritmos, etc , já que apenas com a prática se aprendem certos detalhes da linguagem.

9. COMO ACESSAR O VÍDEO

Link de acesso: **https://youtu.be/c3RttmvVONk**