Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Corretora de Imoveis

BCC221 - Programação Orientada a Objetos

Tiago Mol, Vinicus Anjos, Luiz Eduardo, Leandro Augusto
Professor: Guillermo Cámara-Chávez

Ouro Preto
23 de novembro de 2023

Sumário

Lista de Figuras

1 Introdução

No cenário dinâmico do mercado imobiliário, a eficiência na gestão de imóveis desempenha um papel crucial para o sucesso de uma corretora. Com a evolução das tecnologias de programação, a Standard Template Library (STL) surge como uma ferramenta poderosa, oferecendo uma variedade de contêineres e algoritmos que podem ser empregados de maneira eficaz na implementação de sistemas complexos. Este relatório abordará a criação de uma corretora de imóveis, com foco na utilização criteriosa da STL para a gestão de imóveis e proporcionar uma experiência eficiente para os usuários.

1.1 Objetivo Geral

Desenvolver uma corretora de imóveis com um sistema de gerenciamento eficiente, baseado na Standard Template Library (STL),

1.2 Objetivos Específicos

Filtragem e Classificação de Imóveis: Criar funcionalidades para filtrar imóveis com base em critérios como tipo (casa, apartamento, chacara) e outros parâmetros relevantes. Implementar um sistema de classificação dos imóveis, por exemplo, ordenando-os por valor.

Interação com Diferentes Tipos de Imóveis: Permitir a visualização detalhada e a interação específica com cada tipo de imóvel (casa, apartamento, chacara), utilizando as capacidades de polimorfismo da linguagem de programação.

Apresentação Amigável de Dados: Implementar uma interface de usuário que forneça feedback claro e amigável sobre as ações realizadas, utilizando recursos visuais adequados.

1.3 Justificativa

Optamos pela implementação utilizando exclusivamente o contêiner vector da Standard Template Library (STL) devido à nossa familiaridade com essa estrutura e à percepção de facilidade de manipulação que ela proporciona. A escolha se fundamenta na simplicidade e versatilidade oferecidas pelo vector, tornando a implementação mais acessível e facilitando a compreensão do código. A decisão de se concentrar em um único contêiner visa também a coesão do código, simplificando a manutenção e promovendo uma abordagem mais direta para atingir os objetivos propostos

1.4 Ferramentas utilizadas

- Ambiente de desenvolvimento do código fonte: Vscode.
- Linguagem utilizada: C++.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [1]

1.5 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz.
- Memória RAM: 16Gb.
- Sistema Operacional: Linux.

[1] Disponível em https://www.overleaf.com/

2 Desenvolvimento

O desenvolvimento do programa teve início com a definição clara dos requisitos e funcionalidades esperadas para a corretora de imóveis. Uma análise detalhada das necessidades do sistema orientou a escolha do contêiner vector da Standard Template Library (STL) como a estrutura principal para o gerenciamento dos imóveis. Essa decisão foi motivada pela familiaridade e facilidade percebida na manipulação de dados usando o vector, o que facilitou o desenvolvimento do código.

A implementação seguiu uma abordagem iterativa, começando com a criação das classes necessárias para representar os diferentes tipos de imóveis, como casas, apartamentos e chácaras. A utilização de herança e polimorfismo possibilitou uma estrutura flexível que permitiu tratar os diferentes tipos de imóveis de maneira uniforme quando necessário.

A escolha estratégica da STL se estendeu à aplicação de algoritmos prontos, como a função sort para ordenar os imóveis por valor. Esse elemento da STL contribuiu significativamente para a eficiência do programa.

Durante o desenvolvimento, a interação específica com cada tipo de imóvel foi implementada, utilizando o mecanismo de dynamiccast para realizar verificações de tipo dinâmico. Isso possibilitou a apresentação detalhada e personalizada das informações de cada imóvel.

A interface de usuário foi projetada para fornecer feedback claro sobre as ações realizadas, facilitando a interação do usuário com o sistema. Testes abrangentes foram conduzidos para garantir a correção e eficiência do programa, abordando diferentes cenários e quantidades variáveis de dados.

2.1 Diagrama das Classes

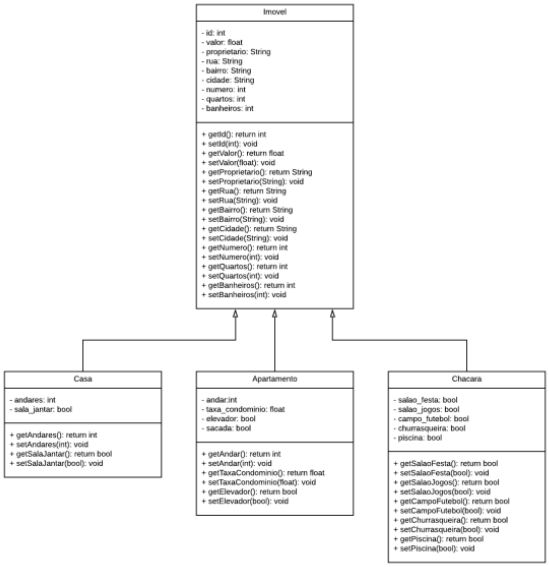

Figura 1: Diagrama

Além das funções definidas no enunciado do trabalho, foi adicionada a função de sobrecarregar o operador <<para todas as classes, além do destrutor virtual na classe de imovel. Todas as outras funções utilizadas foram funções globais.

3 Respostas - Melhores contêiners

3.1 Acessar uma posição específica de um contêiner;

Melhor Contêiner: std::vector ou std::deque. Descrição: Ambos suportam acesso direto a posições específicas e fornecem complexidade de tempo O(1) para essa operação.

3.2 Adicionar um elemento e manter somente elementos únicos no contêiner

Melhor Contêiner: std::set ou std::unorderedset. Descrição: Ambos garantem a unicidade dos elementos, mas o std::unorderedset geralmente tem melhor desempenho para operações de inserção e busca.

3.3 Inserção no final

Melhor Contêiner: std::vector ou std::deque. Descrição: Ambos oferecem inserção eficiente no final, com complexidade de tempo O(1).

3.4 Remoção no final

Melhor Contêiner: std::vector ou std::deque. Descrição: Ambos suportam remoção eficiente no final, com complexidade de tempo O(1).

3.5 Retornar um valor baseado em uma chave específica

Melhor Contêiner: std::map ou std::unorderedmap. Descrição: std::map é ordenado, enquanto std::unorderedmap tem geralmente melhor desempenho para busca, dependendo da situação.

3.6 Inserção no início

Melhor Contêiner: std::list ou std::deque. Descrição: std::list é especialmente eficiente para inserções no início, embora std::deque também seja razoavelmente eficiente.

3.7 Remoção no início

Melhor Contêiner: std::list. Descrição: std::list é eficiente para remoções no início devido à sua estrutura de lista duplamente encadeada

3.8 Busca por um elemento

Melhor Contêiner: std::set ou std::unorderedset. Descrição: Ambos oferecem busca eficiente, mas std::unorderedset geralmente tem um desempenho melhor em cenários de busca.

3.9 Contêiner com o comportamento de primeiro a entrar e o último a sair

Melhor Contêiner: std::stack ou std::deque. Descrição: Ambos são adequados para um comportamento de pilha (LIFO), mas std::stack é uma interface específica para pilha e std::deque também pode ser usado com eficiência.

3.10 Contêiner com o comportamento de primeiro a entrar e o primeiro a sair

Melhor Contêiner: std::queue ou std::deque. Descrição: Ambos são adequados para um comportamento de fila (FIFO), mas std::queue é uma interface específica para fila e std::deque também pode ser usado com eficiência.

4 Conclusão

Na conclusão, destaco que o sistema proposto representa uma solução abrangente e polimórfica para a gestão de imóveis, abordando três tipos distintos: casa, apartamento e chácara. A utilização de um único contêiner polimórfico para armazenar objetos desses tipos permite uma abordagem coesa e eficiente na manipulação dos dados, proporcionando uma estrutura flexível e fácil de entender.

Ao adotar a abordagem polimórfica, o sistema beneficia-se da capacidade de tratar diferentes tipos de imóveis de forma uniforme, simplificando a implementação e promovendo a reutilização de código. As operações definidas como funções são modularizadas, favorecendo a manutenção do código e facilitando a extensão do sistema para incluir novas funcionalidades no futuro.

Em resumo, o sistema proposto não apenas atende aos requisitos funcionais, mas também se destaca pela simplicidade, modularidade e eficiência na gestão e manipulação de diferentes tipos de imóveis. A implementação utilizando a abordagem polimórfica reflete as melhores práticas de desenvolvimento, proporcionando um sistema robusto e adaptável para a gestão eficiente do trabalho proposto.