UNIVERSIDADE FEDERAL DE OURO PRETO
DEPARTAMENTO DE COMPUTAÇÃO

Iago de Castro Andrade - 20.1.4011
Guilherme Augusto Anício Drummond do Nascimento - 20.1.4007
Lorrayne Cristine Ferreira Santos - 20.1.4009
Lucas Silva Barbosa - 20.2.4019

PROGRAMAÇÃO ORIENTADA A OBJETOS
TRABALHO PRÁTICO I

OURO PRETO – MG
2021

1. INTRODUÇÃO

Sabe-se que a programação orientada a objetos é um paradigma que se baseia no conceito de classes e objetos . É usado para estruturar um programa de software em pedaços simples e reutilizáveis. De forma a tornar os projetos de código mais eficientes para manutenção e desenvolvimento colaborativo.

Para a realização do trabalho prático foram usados tais conceitos de desenvolvimento, juntamente com os fundamentos da *Standard Template Library* (STL).

A *Standard Template Library* 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.

A STL do C++ é composta por três componentes, sendo eles: algoritmos, containers e iteradores.

Containers

Os containers são usados para gerenciar coleções de objetos de um determinado tipo. Existem vários tipos diferentes de recipientes, como deque, lista, vetor, mapa, etc.

Algoritmos

Algoritmos atuam em containers. Eles fornecem os meios pelos quais se executa a inicialização, classificação, pesquisa e transformação do conteúdo dos containers.

Iteradores

Iteradores são usados para percorrer os elementos de coleções de objetos. Essas coleções podem ser containers ou subconjuntos de containers.

Para executar o programa, basta executar o seguinte comando baseado no sistema operacional de sua máquina:

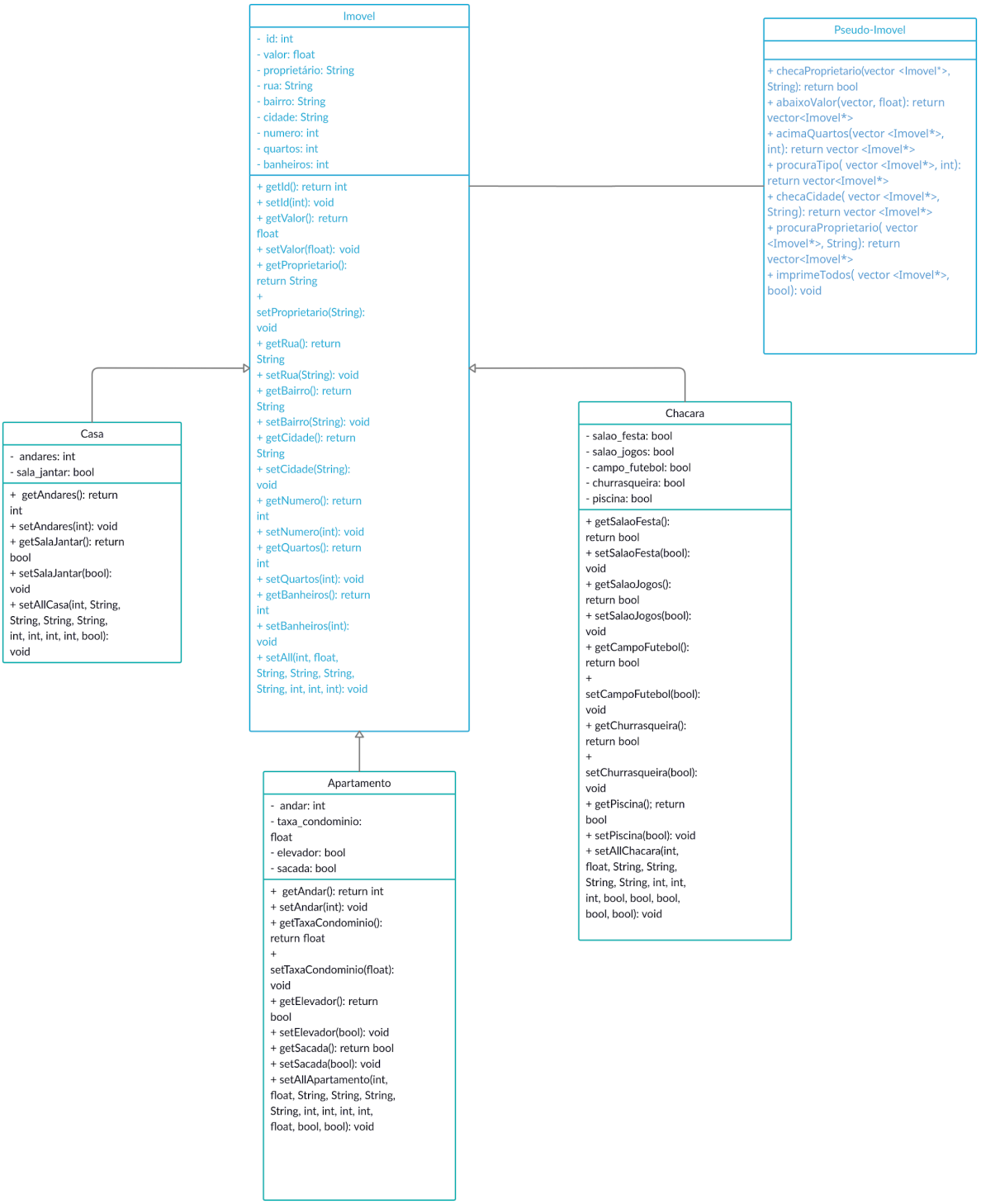
WINDOWS

```
> g++ apartamento.cpp casa.cpp chacara.cpp imovel.cpp main.cpp -o
programa.exe -Wall
> ./programa.exe
```

LINUX

```
$ g++ apartamento.cpp casa.cpp chacara.cpp imovel.cpp main.cpp -o
programa -Wall
$./programa
```

Foi construído um diagrama UML para ser utilizado como base para a implementação do código:



2. ANÁLISE DA IMPLEMENTAÇÃO

2.1. Containers e sua implementação

Os elementos de um vetor são colocados em um armazenamento contíguo para que possam ser acessados e percorridos usando iteradores. O acesso pode ser realizado de forma direta, sendo este o motivo da escolha do mesmo.

```
vector<string> informacoes;
```

Além de tais fatores, ocorre o uso do *downcasting* no qual consiste em converter o ponteiro da classe base (ou referência) em um ponteiro da classe derivada. Sendo tal método utilizado por decorrência da herança no código.

```
vector<Imovel*> imoveis;
```

Tendo sido escolhido pelo mesmo motivo de implementação dos vectores anteriores, o padrão de implementação do vetor a seguir é utilizado em várias funções e exerce o mesmo propósito. De tal forma, o vetor é preenchido contendo as informações necessárias referentes a cada função e assim tem seu valor retornado. O padrão de implementação do vector e das funções que utilizam-no é respectivamente:

```
vector<Imovel*> saida;
saida.push_back(i);
return saida;
```

```
vector<Imovel*> abaixoValor,
vector<Imovel*> acimaQuartos,
vector<Imovel*> procuraTipo,
vector<Imovel*> checaCidade,
vector<Imovel*> procuraProprietario
```

2.2. Algoritmo e sua implementação

O algoritmo Sort foi o escolhido para a implementação, o mesmo tem como característica a classificação dos elementos no intervalo `[first,last)` em ordem crescente. No entanto, o padrão de ordenação pode ser alterado.

Os elementos são comparados usando o operador ‘<’ ou ‘>’ para a primeira versão e, usa da função de comparação binária fornecida ‘comp’ para a segunda. O mesmo possui complexidade $O(n \log n)$.

Os elementos equivalentes não têm garantia de manter sua ordem relativa original (ver stable_sort). Os parâmetros utilizados no algoritmo são:

- [first,last), sendo a gama de elementos a ser classificada;
- ‘comp’ função de comparação que retorna verdadeiro se o primeiro argumento é menor que o segundo.

Deste modo, mediante a suas características de implementação e boa usabilidade, juntamente ao fato de ter uma implementação transparente e acessível de forma a atender aos requisitos do trabalho prático, tornou-se o algoritmo escolhido.

```
sort(saida.begin(), saida.end(), [](Imovel* im1, Imovel* im2){
return im1->getValor() < im2->getValor();});
//implementação para ordem crescente
```

```
sort(saida.begin(), saida.end(), [](Imovel* im1, Imovel* im2){
return im1->getValor() > im2->getValor();});
//implementação para ordem decrescente
```

3. ANÁLISE DE CONTAINERS

- Acessar uma posição específica de um container:
  - ‘vector’: permite o acesso direto via uso de índice. Ele suporta iteradores de acesso aleatório, que são normalmente implementados como ponteiros para os elementos de um vetor;

- Adicionar um elemento e manter somente elementos únicos no container:
  - ‘set’: armazena um conjunto de chaves sem repetição, são containers que armazenam elementos exclusivos seguindo uma ordem específica. Possui complexidade logarítmica $O(\log n)$.
  - Em um conjunto, o valor de um elemento também o identifica (o próprio valor é a chave, do tipo T), e cada valor deve ser único.

- Inserção no final:
  - ‘vector’: sempre insere um elemento no final da fila;

- Remoção no final:
  - ‘vector’: inserção e remoção de elementos no final dele tem tempo constante.

- Retornar um valor baseado em uma chave específica (não necessariamente inteiros):
  - ‘map’: contém pares de valores-chave com chaves exclusivas, a complexidade da pesquisa para `std::map` é $O(log\, n)$ (logarítmica no tamanho do container).

- Inserção no início:
  - ‘deque’ (fila de duas extremidades): permite rápidas inserções e exclusões no início e no final do container. Eles são semelhantes aos vetores, mas são mais eficientes no caso de inserção e exclusão de elementos.

- Remoção no início:
  - ‘deque’ (fila de duas extremidades): são filas em que as operações de inserção e exclusão são possíveis em ambas as extremidades.
  - As funções para `deque` são iguais às de vetor, com uma adição de operações `push` e `pop` para frente e para trás.

- Busca por um elemento:
  - ‘set’ e ‘map’: realizam a pesquisa em tempo $O(log\, n)$, em comparação com vector e list, que realizam a mesma pesquisa em tempo $O(n)$, ou seja, set e map são melhores para realizar a busca de um elemento;

- Container com o comportamento de primeiro a entrar é o último a sair:

    - `‘stack’`: funciona como uma pilha, ou seja, o último elemento a ser inserido, será o primeiro a ser retirado.

- Container com o comportamento de primeiro a entrar é o primeiro a sair:
    - `‘queue’`: são estruturas de dados do tipo FIFO (first-in, first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.

4. CONCLUSÃO

A STL é uma biblioteca que contém artefatos que possibilitam um código mais limpo e ágil de ser implementado, facilitando o uso da linguagem C++ através da disponibilização de uma implementação eficiente de estruturas de dados com alto nível de abstração.

No entanto, é necessário conhecer acerca dos diferentes métodos de algoritmos, containers e iteradores para a produção de um código mais eficaz, juntamente com com os princípios da programação orientada a objetos (classe, objeto, herança, polimorfismo e encapsulamento).