UNIVERSIDADE FEDERAL DE OURO PRETO
DEPARTAMENTO DE COMPUTAÇÃO - DECOM
Professor: Guillermo Camara Chavez

RELATÓRIO PRÁTICO

Disciplina: BCC221 **Turma:** 11

Alunos:
Artur Bermond Torres (21.1.4003)
Kemuel Marvila (21.1.4013)
Pedro Augusto Sousa Gonçalves (21.1.4015)

Ouro Preto
2022

Introdução

Através de um contexto específico, desenvolveremos uma aplicação que utiliza o paradigma de Programação Orientada a Objetos para trazer certas abstrações natas à arte de programar mais próximas da realidade. Utilizando poderosas ferramentas fornecidas a nós por meio da STL do C++, implementaremos diversas funções para manusear livros e seus diferentes atributos, além de processar tais atributos para obter e organizar informações conforme solicitado.

Desenvolvimento

Primeiramente, vale ressaltar que seguimos o diagrama UML fornecido à risca, com a classe Livro sendo abstrata e as três subclasses sendo derivadas da supracitada.

Dito isso, o diagrama nos deu certa liberdade para escolher o contêiner do STL a ser usado. Nós optamos por usar Vectors em **quase** todo o projeto, por alguns motivos: Primeiro, os arquivos de texto com os livros não nos forneciam qualquer item que agisse como uma chave única, então containers associativos como maps e sets não seriam tão úteis. Utilizar o multiset era uma opção, com o título do livro sendo a chave e portanto abrindo espaço para um mesmo livro ter uma versão digital e física, por exemplo. No entanto, a quantidade de dados fornecidos não faria com que isso fosse algo tão impactante, não nos traria benefícios consideráveis para a implementação do que foi solicitado. É elegante, mas para o nosso objetivo, uma elegância redundante. Portanto, seguindo a filosofia de desenvolvimento KISS (link), decidimos nos manter com o vector. Além disso, o tamanho do nosso conjunto de dados não traria benefícios consideráveis ao uso de estruturas mais esparsas, como lists, pois é fácil alocar 16 posições sequenciais na memória para armazenar os livros, e como não precisamos adicionar mais dados depois de ler os arquivos pela primeira vez, a expansão do vector não será um problema. O acesso sequencial é uma vantagem dessa estrutura, pois isso nos dá uma vantagem de performance. Entendemos que talvez o número de livros fornecidos para a execução do trabalho foi pequeno por motivos didáticos, mas também entendemos que a análise da magnitude dos dados a serem processados é uma habilidade crucial de todo programador, principalmente quando for preciso escolher dentre várias estruturas que o C++ nos proporciona.

Na “g”, no entanto, nós utilizamos um Set, pois ele automaticamente retira qualquer elemento duplicado que possa ser adicionado, o que foi usado para retirar todas as *keywords* duplicadas.

Quanto a algoritmos, utilizamos sempre que possível aquilo já implementado na biblioteca padrão do C++, como na questão “c”, onde usamos a função sort para ordenar os livros pelo

ano. Além disso, iteradores foram usados em quase todas as questões deste trabalho prático, o que serve para mostrar a imensa utilidade que eles oferecem.

Agora, o PDF com as instruções para o trabalho nos perguntou qual contêiner STL é o melhor para cada um dos objetivos abaixo. Eis os casos e as respostas:

- Acessar uma posição específica de um contêiner
    - Array, porque permite acesso baseado em index.
- Adicionar um elemento e manter somente elementos únicos no contêiner
    - Set, porque ignora inserções de elementos repetidos.
- Inserção no final
    - List, Stack, Queue ou Deque. **(*)**
- Remoção no final
    - List, Stack ou Deque. **(*)**
- Retornar um valor baseado em uma chave específica (não necessariamente inteiros)
    - Map, porque relaciona chaves únicas com valores.
- Inserção no início
    - List ou Deque. **(*)**
- Remoção no início
    - List, Queue e Deque. **(*)**
- Busca por um elemento
    - Containers associativos em geral, como map, set, multimap ou multiset, pois a busca nesses é O(log n).
- Contêiner com o comportamento de primeiro a entrar é o último a sair
    - Stack, pois segue o padrão LIFO
- Contêiner com o comportamento de primeiro a entrar é o primeiro a sair
    - Queue, pois segue o padrão FIFO.

(*) Pois tal operação é O(1) sempre para essas estruturas.

Conclusão

Não é controverso dizer que a prática nos dá um conhecimento muito valioso. Com essa prática, tivemos que discutir qual estrutura de dados STL usar, com base em um problema fornecido. A escolha é difícil, e inclusive há discordância no grupo sobre qual seria a melhor para esse tipo de problema, mas com uma boa comunicação entre os integrantes do grupo e um pensamento claro sobre pontos positivos e negativos de cada uma, nós julgamos a nossa escolha como satisfatória para o *dataset* do enunciado. POO nos oferece uma nova lente sobre programação, levando a resolução de problemas para a vida real.