Simulação para Fluxo de Containers de Carga Utilizando C++

Letícia Luiza Cardoso
leticialuizac@gmail.com
UFSC Joinville
Santa Catarina, Brasil

Valquiria Rafaela Radunz
valrradunz@gmail.com
UFSC Joinville
Santa Catarina, Brasil

***Resumo* — O objetivo deste trabalho é demonstrar como foi executada a criação de um código na linguagem de programação C++, cuja funcionalidade envolve o gerenciamento de containers de carga. As informações contidas neste trabalho exemplificam o desenvolvimento e a execução de um sistema de cadastro desenvolvido com a Programação Orientada a Objetos (POO).**

Palavras-chave — programação, containers, C++, POO .

I. INTRODUÇÃO

O código foi desenvolvido para realizar funções básicas de cadastro de containers de carga, simulando chegada e saída em um porto seco ou Estação Aduaneira do Interior (EADI). Dentro da linguagem de programação C++, foi utilizado o conceito de programação orientada a objetos para auxiliar na eficiência do código.

No programa desenvolvido, os objetos principais são os containers e as datas utilizadas para o cadastro destes containers no sistema. Primeiramente o usuário insere informações do container em questão, e em seguida acontece a inserção da data de saída, para que o sistema analise o tempo de estadia em relação a data atual, e defina onde o container será colocado de acordo com seu nível de prioridade. Por fim, os containers são retirados do pátio quando chega a data de saída definida pelo usuário.

II. DESENVOLVIMENTO

A ideia do programa surgiu para relacionar o aprendizado sobre a linguagem de programação C++, e os conteúdos estudados em outras disciplinas do curso de Engenharia de Transportes e Logística, ofertado pela Universidade Federal de Santa Catarina no *campus* de Joinville. O programa foi desenvolvido para ser uma simulação de sistema de cadastro de containers para armazenamento em um pátio (porto seco ou EADI), e o usuário pode inserir as informações do container, sua data de entrada e de saída.

A. Classe Container

De acordo com os conceitos de POO, os containers foram definidos como objetos principais do programa para facilitar sua execução. A classe Container foi criada com atributos de identificação, de acordo com padrões reais estabelecidos pela Organização Internacional de Normalização (ISO).

```
#pragma once

#include <iostream>
#include <string>
#include "data.h"

using std::string;

class Container{

public:
    //construtores
    Container(); //construto padrão

    Container(string cod){
        _tam = 20;
        _codigo = cod;
    }

    std::string getCodigo();
    int getTamanho();
    void setTamanho(int t);
    int getPrioridade();
    void setPrioridade(int prioridade);

    Data _dataSaida;
    Data _dataEntrada;

    ~Container();

private:
    //atributos

    int _prioridade; // Diferenca de dias entre a data atual e a data de saida
    string _codigo;
    int _tam;        //20 ou 40 pés
};
// #endif
```

Figura 1 - Classe Container

O código de identificação de um container é formato por 11 elementos alfanuméricos:

1) Código do proprietário: Composto por 3 letras do alfabeto referentes ao código de registro ISO do proprietário, e 1 letra que sempre será “U” indicando “Unit” ou “unidade”.

2) Número de série: Formado por 6 algarismos romanos, que indicam o número do container em relação a sua frota.

3) Dígito de controle: Número único para cada container, obtido através de uma operação matemática que envolve o valor predeterminado das letras (determinado pelo ISO), e os números de série. A seguir, uma exemplificação do cálculo:

a) Valores para cada letra: Para cada letra do alfabeto, em ordem, os valores respectivos são: 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38.

b) Valores para os algorismo romanos: Os números permanecem com seu valore padrão.

*c) Código de exemplo **AKVY751013**:* Para cada letra e número, multiplicamos o valor respectivo por 2 elevado ao valor de sua posição. O cálculo de A pode ser visto em (1), onde 10 é o valor padrão para a letra A, e 0 é relativo à primeira posição dos elementos. Em (2) o valor do número 7 é multiplicado por 2 elevado na potência 4, que representa a quinta posição dos elementos alfanuméricos.

$$\text{A: } 10 \times 2^0 = 10 \qquad (1)$$

$$\text{7: } 7 \times 2^4 = 112 \qquad (2)$$

Feito o cálculo para cada posição, somam-se os resultados e o valor obtido é divido por 11. No caso do exemplo, obteve-se 2612 como resultado da soma dos cálculos das posições, e a operação de divisão pode ser vista em (3).

$$2612 \div 11 = 237{,}45 \qquad (3)$$

Em seguida utiliza-se a parte inteira do resultado da divisão e multiplica-se por 11, de acordo com (4).

$$237 \times 11 = 2067 \qquad (4)$$

O dígito de controle é obtido através de (5), quando o último valor encontrado é subtraído da soma das posições.

$$2612 - 2067 = 5 \qquad (5)$$

Portanto o código de exemplo estará completo com o dígito 5 (**AKVY751013-5**).

Além disso, a classe Container exige que o objeto possua tamanho para que não sejam empilhados containers de diferentes tamanhos, e estes são dois valores padrões determinados pelo ISO, que possuem dois tipos distintos de pilhas no pátio: para 20 ou 40 pés de comprimento. Por fim, a classe Container recebe sua data de saída, e partir dessa informação define a prioridade do objeto: quanto maior o período de permanência do container, menor sua prioridade na pilha. Dessa forma, a alocação na pilha permite que os containers que devem sair antes estejam em cima dos que precisam sair depois.

B. Classe Pilha

A classe Pilha, criada com o auxílio da biblioteca padrão *stack*, é responsável por guardar os containers no pátio. Nesta simulação, definiu-se que existem 10 pilhas, 5 para cada tamanho de container, nas quais pode-se empilhar até 5 containers. A classe verifica o tamanho de cada container para empilhar os de comprimentos iguais, e quantos containers já foram inseridos para confirmar se existe espaço para mais um. Além disso, é responsabilidade desta classe inserir ou remover containers das pilhas.

```
#pragma once

//  #ifndef PILHA_H
// #define PILHA_H

#include <iostream>
#include <stack>
#include "container.h"

#define TAMANHO_MAXIMO_PILHA 5

class Pilha{

public:
    Pilha();// Construtor
    Pilha(int tam);// Construtor com inicialização do comprimento dos containers
    Pilha(Container *); // Construtor com inserção de container
    ~Pilha();// Destrutor

    // Métodos //

    // Insere um conteiner na pilha
    // Retorna 1 se inseriu com sucesso ou 0 em caso de falha
    int insereContainer(Container * container);

    // Remove um conteiner na pilha
    // Retorna o container que foi removido
    // Se a pilha estiver vazia retorna nullptr
    Container* removeContainer();

    int getTamMax();

    int getNumeroContainers();

    int getComprimentoContainers();



private:

    //Atributos
    int _tamMaxPilha = TAMANHO_MAXIMO_PILHA;    // Numero maximo de container na pilha
    int _nContainers;                           // Numero atual de containers
    int _tamContainer;                          // Comprimento dos containers da pilha (20 ou 40 pés)
    std::stack<Container*> * _pilha;            // Pilha de containers
};
// #endif
```

Figura 2 - Classe Pilha

C. Classe Setor

A classe Setor é responsável pelo agrupamento das pilhas, o qual facilita a localização dos containers e suas respectivas prioridades. No código em questão, são criados dois setores distintos para os dois tamanhos possíveis de containers.

```
#pragma once

// #ifndef SETOR_H
// #define SETOR_H

#include <iostream>
#include <stack>
#include "pilha.h"

#define TAMANHO_MAXIMO_SETOR 10

class Setor{
public:
    Setor();
    ~Setor();

//private:
    //Atributos//
    Pilha * _setor[TAMANHO_MAXIMO_SETOR];


};

// #endif
```

Figura 3 - Classe Setor

D. Classe Data

Para gerenciar a entrada e a saída dos containers no porto seco, são necessárias informações sobre o tempo de permanência. A classe Data foi criada para auxiliar nessa contagem dentro da simulação, e utiliza a data de entrada (obtida da data atual no computador de operação) e a data de saída para calcular o período de permanência dos containers no pátio. A classe Data também é responsável pela verificação da validade da data de saída inserida, para que não ocorram erros de execução no programa.

```
#pragma once

#include <string>

using std::string;

class Data{
public:
    Data(){
        // Construtor padrao
        dia = 1;
        mes = 1;
        ano = 2020;
    }
    Data(int d, int m, int a){
        // Salva os argumentos passados nos atributos para inicializacao
        setData(d,m,a);
    }

    int getDia();
    int getMes();
    int getAno();

    bool setData(int d, int m, int a);
    bool setData(string data); // sobrecarga para setar a data apartir de uma string do tipo dd/mm/aaaa

    void printData();

    ~Data(){} //dest

    int operator-(Data data);
    Data operator++( int );

private:
    int dia, mes, ano;

};
```

Figura 4 - Classe Data

E. Classe Interface

Para facilitar a visualização do usuário, optou-se por criar um menu impresso em tela com informações sobre as entradas do código. Essa interface foi feita a partir de uma classe utilizando o próprio conceito de POO, de forma que as outras classes podem acessar e alterar o conteúdo impresso em tela através de relações de associação.

```
#pragma once

#include "setor.h"
#include "data.h"
#include <string>
using std::string;

class Interface
{
public:
    void mostraMenu(int c);
    void mostraLog();
    void mostraPilhas();
    ~Interface(){}
    Data diaAtual;

    string log;

    Setor * lote;
};
```

Figura 5 - Classe Interface

III. Discussão

O código de simulação para fluxo de containers começa com o menu inicial e instruções para inserção de dados válidos. Para que a execução comece, o programa exige que seja inserido um código de identificação de container válido, e usuário pode utilizar um código próprio, ou selecionar comandos para teste dentro do programa.

O menu mostra as opções de comandos “v” para gerar um código aleatório válido, “i” para gerar um código aleatório inválido, ou “sair” para fechar o menu do programa e cancelar a execução.

Pode ser vista também a data atual, obtida do computador onde o código está sendo executado, e o setor de pilhas onde serão armazenados os containers. Existe o setor para container de 20 pés, onde as pilhas são enumeradas de 0 a 4, e o setor para containers de 40 pés, com pilhas de número 5 até 9. Na visualização inicial todas as pilhas estão vazias (existe um espaço em branco entre os símbolos “+”). deve ser interpretada da seguinte forma: se o espaço está em branco, aquele andar da pilha está livre.

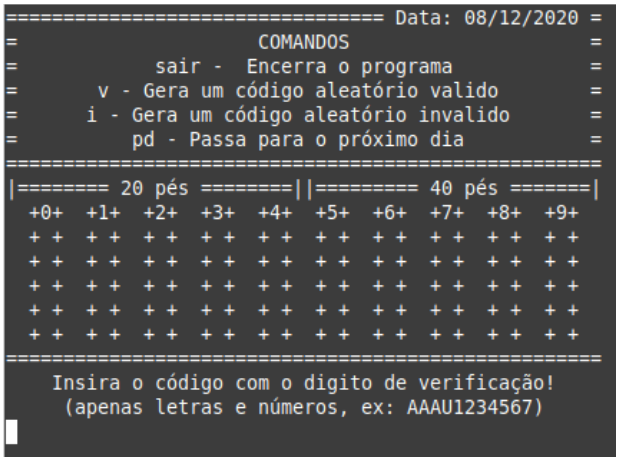

Figura 6 - Menu Inicial

A verificação do código ocorre e retorna a informação de que ele é válido ou não. Se a identificação do container estiver correta, o próximo passo é inserir sua data de saída. Com essa informação, o código valida se a data foi inserida no formato certo, calcula quantos dias serão no total com o auxílio da sobrecarga do operador “-” para fazer a subtração das datas, e define a prioridade do container. A fim de facilitar a inserção de dados para testes, foi criado o comando “a”, o qual utiliza a sobrecarga do operador “++” para somar mais um dia, e define a data de saída como o próximo dia após o dia atual (amanhã).

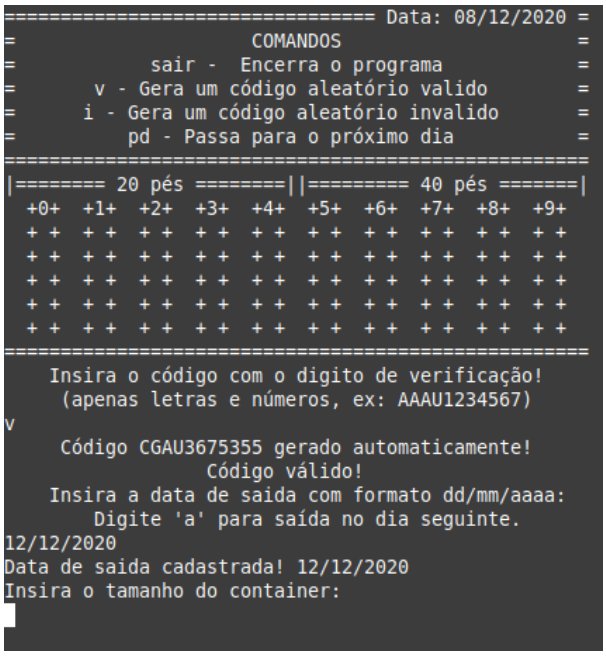

Figura 7 - Inserção de Data

Em seguida, o usuário deve digitar o tamanho do container sem a unidade de medida: 20 ou 40. Com essa informação, o sistema define em qual setor o container deve ser colocado, e depois define em qual pilha ele será inserido. Containers que ficarão por um período mais longo são armazenados na base das pilhas, e em cima deles podem ser empilhados apenas aqueles que possuem prioridade maior ou igual (quanto menor o número atribuído à variável prioridade, menos tempo o container vai ficar

no porto seco e maior sua prioridade) e sairão antes do container que está embaixo. Se uma pilha estiver vazia não existe restrição de prioridade, e qualquer container pode ser colocado. Quando a inserção é executada, o espaço em branco da pilha onde o container foi posicionado recebe o símbolo "=" para ilustrar que foi preenchido.

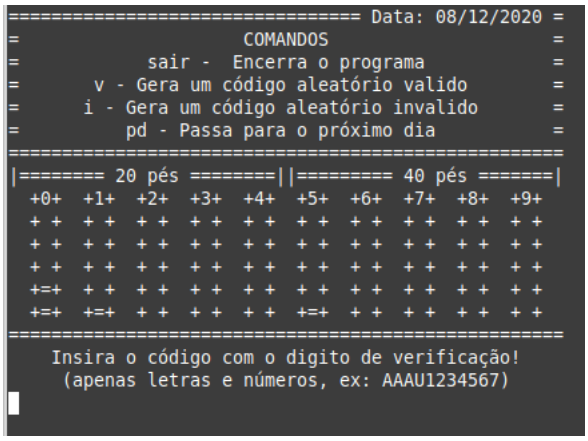

Figura 8 - Containers Inseridos

A última ação a ser realizada é a remoção dos containers. No menu principal é possível passar a data para o próximo dia utilizando o comando "pd", o que faz com que as prioridades de todos os containers diminuam em 1, e, caso algum container fique com prioridade 0, será removido do pátio.

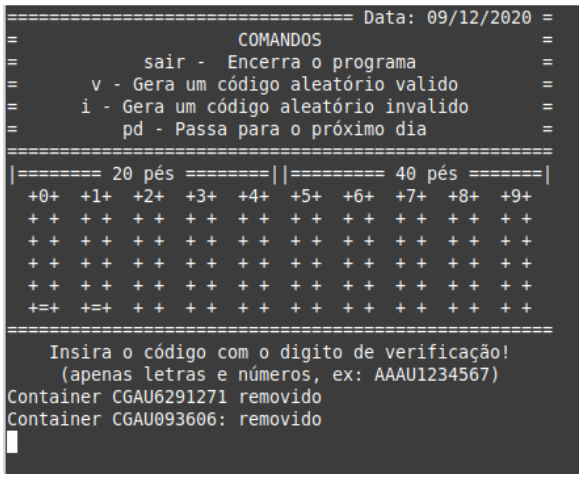

Figura 9 - Containers Removidos

IV. Conclusão

Após a utilização do conceito de POO, percebeu-se que muitas coisas podem ser feitas com a linguagem C++. O resultado final do sistema de simulação almejado foi obtido com certa facilidade em alguns aspectos, como o uso da ferramenta *class* para criar uma data como objeto, que favoreceu a configuração do tempo de permanência dos containers através da aplicação de sobrecarga de operadores. A interface gráfica também desenvolvida foi elaborada exclusivamente com o uso dos conceitos de POO, fato que confirma como a programação orientada a objetos pode ser útil.

Contudo, pôde-se notar que nem todos os conceitos de POO são facilitadores na construção de um código, pois em algumas situações eles poderiam complicar ao invés de simplificar o programa. Devido a isso, optou-se por não utilizar algumas ferramentas de POO na realização deste projeto.

Referências

[1] DEITEL, P.; DEITEL, H. C++: How To Program, 9a edição, Ed. Pearson, 2014. ISBN-10: 0133378713.

[2] SAVITCH, W. J.. C++ Absoluto. São Paulo: Addison Wesley. 2004. ISBN: 85-88639-09-2.

[3] O que é e como funciona um porto seco. Remessa Online, 2020. Disponível em:< https://www.remessaonline.com.br/blog/o-que-e-e-como-funciona-um-porto-seco/ >. Acesso em: 26 de nov. de 2020.

[4] ISO 6346:1995 Freight containers — Coding, identification and marking. ISO, 1995. Disponível em:< https://www.iso.org/obp/ui/#iso:std:iso:6346:ed-3:v1:en >. Acesso em: 30 de nov. de 2020.

[5] Container Identification Number. Bureau International des Containers et du Transport Intermodal (BIC). Disponível em:< https://www.bic-code.org/identification-number/ >. Acesso em: 30 de nov. de 2020.