UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Segundo Trabalho de Programação III

Relatório

Matheus Gomes Arante de Souza
Matheus Salomão

Vitória
Novembro de 2018

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Matheus Gomes Arante de Souza
Matheus Salomão

Segundo Trabalho de Programação III

Relatório

Trabalho referente à disciplina de Programação III do curso de Ciência de Computação do Departamento de Informática da Universidade Federal do Espírito Santo.

Vitória
Novembro de 2018

Sumário

1 Introdução

Esse trabalho tem como principal objetivo pôr em prática os conhecimentos adquiridos em sala de aula sobre programação orientada a objetos, mais especificamente a linguagem de programação C++.

Dado um arquivo de entrada com os resultados de uma eleição municipal, o código visa armazenar e analisar essas variáveis retornando na saída padrão os seguintes relatórios:

- Número de vagas;
- Candidatos eleitos;
- Candidatos mais votados dentro do número de vagas;
- Candidatos não eleitos e que seriam eleitos se a votação fosse majoritária;
- Candidatos eleitos no sistema proporcional vigente, e que não seriam eleitos se a votação fosse majoritária;
- Votos totalizados por coligação ou partido;
- Votos totalizados por partido;
- Total de votos nominais.

Cada um dos relatórios estará ordenado a partir do número de votos contabilizados pelo item em questão e cada um destes apresentará informações consideradas pertinentes, por exemplo, junto ao vereador estará vinculado seu partido, coligação e número de votos.

Este relatório tem como objetivo apresentar cada passo do código, explicando cada classe e função criada, além das abordagens escolhidas e os problemas e dificuldades que ocorreram durante o processo de desenvolvimento do trabalho.

O processo de explicação do código será desenvolvido em etapas. Cada classe será divida em um capítulo específico da seção de Implementação das Classes do Projeto[2] e dentro de cada capítulo serão explicadas as funções e atributos referentes a classe.

2 Implementação das Classes do Projeto

2.1 Candidato

2.1.1 Atributos

A classe de Candidato possui os seguintes atributos:

- o nome do candidato;
- a referência para o partido que ele pertence;
- o número de votos recebidos;
- e o campo eleito.

Cada um desses atributos possui sua função getter e setter correspondente.

2.1.2 Função comparaCandidatos

A função comparaCandidatos recebe como parâmetro dois candidatos e verifica se o número de votos recebidos do primeiro é maior do que o número de votos do segundo e retorna um valor booleano:

- true, caso o número de votos do primeiro for maior que do segundo;
- false, caso os valores sejam iguais ou o número de votos do primeiro seja menor que o do segundo.

Essa função foi criada para que a lista de vereadores possa ser ordenada posteriormente, a fim de plotar os resultados de forma decrescente de número de votos.

2.1.3 Função ToString

Essa função é responsável por exibir na saída padrão as informações necessárias sobre o candidato. Essa função possui um if-else para controlar o número de informações passadas. Caso o candidato não esteja em uma coligação com mais de um partido não é necessário informar a coligação de que ele pertence pois isso representaria apenas o seu próprio partido.

As ideias que giram em torno da abordagem de um partido sem coligação ser encaixado em uma coligação com apenas um partido será abordada no capítulo referente a classe Coligação [2.3].

2.2 Partido

2.2.1 Atributos

A classe Partido possui os seguintes atributos:

- o nome do partido;
- a lista de candidatos afiliados ao partido;
- a referência para a coligação do partido;
- o número de votos que o partido recebeu (soma dos votos dos candidatos).

O nome do partido, sua coligação e a lista de candidatos possuem suas respectivas funções getter e setter, além de uma função insere para adicionar um novo vereador a lista.

2.2.2 Função votosTotais

Essa função consiste em somar o número de votos recebidos por cada candidato afiliado ao partido, isto é, da lista de candidatos do partido e retornar o resultado total.

2.2.3 Função numCandidatosEleitos

Mais uma vez acessando a lista de candidatos do partido, essa função verifica se o campo eleito do candidato é verdadeiro, caso seja, uma variável auxiliar é incrementada. Ao final do loop é possível saber o número de candidatos eleitos que pertencem àquele partido. Esse valor é retornado pela função.

2.2.4 Função dadosDoPartido

O nome do partido, número total de votos contabilizados e o número de candidatos eleitos é exibido na saída padrão. Além disso, existe um if-else no final dessa função para determinar a flexão de número do substantivo "eleito", que complementará o texto exibido na saída padrão.

2.2.5 Função comparaPartidos

Assim como a classe Candidato, esta classe também possui uma função que compara dois objetos, no caso partidos, a partir do seu número de votos contabilizados. A comparação se através do mesmo método, ou seja, uma comparação que pode retornar true ou false, cada um com seu respectivo significado. Para mais informações verifique a função comparaCandidatos da classe Candidato [2.1].

2.3 Coligação

2.3.1 Atributos

A classe Coligação possui os seguintes atributos:

- uma lista de partidos;
- o número de votos recebidos.

Relacionado a essa lista existe uma função que retorna uma cópia dessa lista, denominada getPartidos, uma função insere que adiciona um novo partido a coligação e uma função contem que verifica se um partido está contido na lista de partidos da coligação.

2.3.2 Função ToString

Essa função é responsável por criar uma String contendo o nome de cada partido pertencente a coligação. Seguindo o mesmo padrão do arquivo de entrada, cada partido é separado por uma barra.

2.3.3 Função votosTotais

Através de um loop, os votos de cada partido filiado a coligação são somados e o resultado final, que representa o número de votos totais contabilizados pela coligação são retornados.

2.3.4 Função dadosDaColigacao

Inicialmente, o número de votos totais recebidos e o número de candidatos eleitos pela coligação são contados e adicionados a variáveis. Através da função ToString, os nomes dos partidos coligados são exibidos na saída padrão, juntamente com o número de votos e candidatos eleitos desta coligação.

A função possui um if-else para determinar a flexão de número correta ao substantivo relacionado ao número de candidatos eleitos.

2.3.5 Função comparaColigacoes

A coligação também possui uma função de comparação própria que determina qual coligação é "maior"ou "menor"a partir do número de votos. Essa função segue o mesmo raciocínio das funções de comparação da classe Candidato [2.1] e Partido [2.2].

Para mais informações verifique a função comparaCandidatos no capítulo sobre a classe Candidato [2.1].

2.4 Eleição

2.4.1 Atributos

A classe Eleição possui os seguintes atributos:

- uma lista de vereadores;
- uma lista de partidos;
- uma lista de coligações;
- o número de votos da eleição.

Cada uma das listas possui seus respectivos getters e setters.

2.4.2 Construtor

O construtor de uma eleição recebe a lista de candidatos, a lista de partidos e a lista de coligações. Essas listas foram ordenadas em ordem decrescente através das funções de comparação que foram definidas para cada uma dessas classes.

2.4.3 Função numVagas

Essa função é responsável por contar o número total de vagas da eleição. Considerando o número de vagas inicialmente como zero, a lista de vereadores é percorrida e caso o candidato em questão tenha sido eleito o número de vagas é incrementado. Ao final desse processo o valor é retornado pela função.

2.4.4 Função totalVotosNominais

Através de um loop, a lista de vereadores é percorrida por completo para somar todos votos e determinar o total de votos contabilizados pela eleição.

2.5 Input/Output

2.5.1 Função LerArquivo

Dado um arquivo de entrada, a função ignora o cabeçalho, isto é, a primeira linha do arquivo, e analisa cada linha individualmente. As linhas são divididas em campos que estão separados pelo delimitador ";".

A partir da leitura dos campos é definido o nome do candidato, seu partido e coligação, o número de votos recebidos e se ele foi eleito ou não. Demais informações são ignoradas. Com esses dados são geradas listas de candidatos, partidos e coligações, e estas são adicionadas a uma eleição [2.4], cuja referência é retornada ao final da função. Caso haja algum problema com a leitura do arquivo, o retorno será null.

2.5.2 Função dadosDosPartidos

Essa função tem como objetivo reunir as informações necessárias para plotar a lista de votação (nominal) dos partidos, apresentando também o número de candidatos eleitos pelo partido.

Através de um loop a lista de partidos é percorrida. A cada iteração, um partido é selecionado, a função dadosDoPartido da classe Partido [2.2] é chamada, retornando portanto o nome do partido, o número de votos recebidos e candidatos eleitos. E além disso, o número de votos é incrementado a uma variável denominada votos_totais, inicialmente setada com zero, que no final da função retornará o total de votos contabilizados durante a eleição.

É importante mencionar que caso o partido não possua candidatos, ele não será apresentado na lista de partidos.

2.5.3 Função dadosDasColigacoes

Essa função é responsável por agrupar dados para plotar a lista de votação (nominal) das coligações e numero de candidatos eleitos em cada coligação.

Percorrendo a lista de coligações, as informações de cada uma delas são exibidas na saída padrão.

2.5.4 Função candidatosEleitos

Essa função é responsável por reunir informações sobre os candidatos que foram eleitos e exibir na saída padrão esses dados.

2.5.5 Função candidatosMaisVotados

A partir da lista de candidatos, serão exibidas na saída padrão informações sobre os candidatos mais votados, em ordem decrescente de votação e respeitando numero de vagas.

2.5.6 Função votacaoMajoritaria

Essa função reúne informações sobre os candidatos que teriam sido eleitos se a votação fosse majoritária e exibe na saída padrão os dados destes candidatos.

2.5.7 Função beneficiadosSistProp

Essa função consiste em reunir informações sobre os candidatos que se beneficiaram do sistema proporcional (com sua posição no ranking de mais votados).

A lista de candidatos, que está ordenada de forma decrescente de número de votos, é percorrida a partir da posição num-vagas+1, caso o candidato tenha sido eleito, o toString dele é chamado, representando portanto, que ele se beneficiou do sistema proporcional.

2.6 Testador

2.6.1 Main

A função main do código cria uma eleição que recebe os dados retornados pela função LerArquivo, e partir disso, ela chama as funções responsáveis por gerar os relatórios especificados na introdução do relatório [1].

3 Conclusão

Os testes do programa foram realizados com a utilização do script de testes disponibilizado no site da disciplina.

```
2017100207@labgrad:~/Downloads/v15/2018-02-trab2$ ./test.sh
Script de teste PROG3 - Trabalho 2

[I] Testando Vereadores...
[I] Testando Vereadores: teste capela
[I] Testando Vereadores: teste capela, tudo OK
[I] Testando Vereadores: teste manaus
[I] Testando Vereadores: teste manaus, tudo OK
[I] Testando Vereadores: teste riodejaneiro
[I] Testando Vereadores: teste riodejaneiro, tudo OK
[I] Testando Vereadores: teste saopaulo
[I] Testando Vereadores: teste saopaulo, tudo OK
[I] Testando Vereadores: teste vilavelha
[I] Testando Vereadores: teste vilavelha, tudo OK
[I] Testando Vereadores: teste vitoria
[I] Testando Vereadores: teste vitoria, tudo OK
[I] Testando Vereadores: pronto!
```

Figura 1: Execução do script de testes no terminal.

Além disso, com o auxílio do utilitário `valgrind`, foi possível verificar questões relacionadas à memória.

```
Total de votos nominais: 4459144

==3148==
==3148== HEAP SUMMARY:
==3148==     in use at exit: 72,704 bytes in 1 blocks
==3148==   total heap usage: 164,760 allocs, 164,759 frees, 42,870,942 bytes allocated
==3148==
==3148== LEAK SUMMARY:
==3148==    definitely lost: 0 bytes in 0 blocks
==3148==    indirectly lost: 0 bytes in 0 blocks
==3148==      possibly lost: 0 bytes in 0 blocks
==3148==    still reachable: 72,704 bytes in 1 blocks
==3148==         suppressed: 0 bytes in 0 blocks
==3148== Rerun with --leak-check=full to see details of leaked memory
==3148==
==3148== For counts of detected and suppressed errors, rerun with: -v
==3148== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 2: Execução do `valgrind` para o divulga.csv referente a cidade de São Paulo.

Visto o feedback do primeiro trabalho, realizamos algumas mudanças estruturais no código, como por exemplo a criação de uma classe para separar a entrada/saída das informações do domínio do problema.

Por meio deste trabalho foi possível colocar em prática diversos conceitos apresentados em sala de aula sobre a linguagem C++.

4 Referências

- https://nemo.inf.ufes.br/jpalmeida/ensino/2018-02-prog-iii/