

UNIVERSIDADE ESTADUAL DE MONTES CLAROS
Centro de Ciências Exatas e Tecnológicas
Curso de Engenharia de Sistemas

Diogo Nascimento Silva
Lucas Franklin Silva

TRABALHO PRÁTICO IV
PROGRAMAÇÃO ORIENTADA A OBJETOS

Montes Claros - MG
Dezembro / 2013

1 – INTRODUÇÃO

Esse trabalho teve como objetivo a aplicação de conceitos de orientação a objetos aprendidos em aula usando a linguagem C++. Neste trabalho usamos alguns dos códigos desenvolvidos desde o primeiro trabalho prático e aplicamos o conceito de herança, composição, encapsulamento, polimorfismo, classes abstratas, além de alguns mecanismos ou padrões como templates, iterator e lançamento de exceções. Pelo fato de termos estudados templates, iterators e lançamento de exceções por conta própria, usando como referencia o livro adotado na disciplina e a internet, alguns códigos podem estar fora dos padrões convencionais de desenvolvimento.

2 – DESENVOLVIMENTO

2.1 - PADRÕES

Foram usados os mesmos padrões de desenvolvimento adotados nos trabalhos anteriores.

2.2 – COMENTÁRIOS

Os comentários dos códigos foram feitos quando achamos realmente necessário, a fim de economizar tempo. Sendo assim, algumas coisas importantes podem ter passado despercebidas.

2.3 – TESTES E IMPLEMENTAÇÃO

Abaixo serão listados alguns resultados e discussões importantes que tivemos em cada questão:

2.3.1 – PROBLEMA 2.1

2.3.1.1 – Modelagem

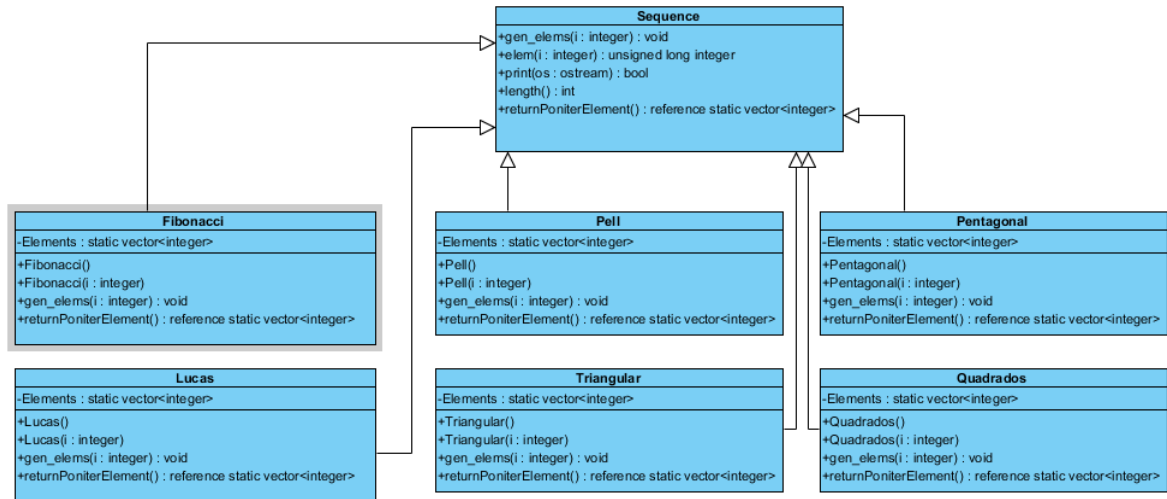
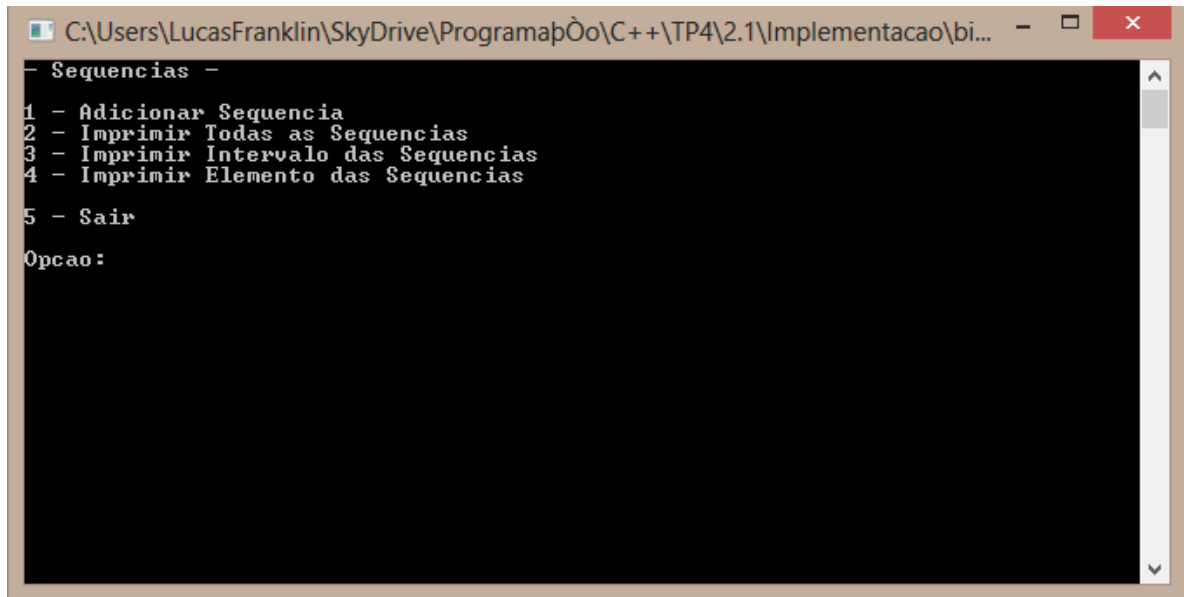


Figura1: Modelo da Classe

Foram modelados os métodos básicos do problema. Tentamos deixar o mais genérico possível, de forma que possa ser aplicado em qualquer linguagem de programação. Sendo assim, a sobrecarga do operador de fluxo de saída não consta no modelo.

2.3.1.2 - Implementação




```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP4\2.1\Implementacao\bi... - [ ] [X]
- Sequencias -
1 - Adicionar Sequencia
2 - Imprimir Todas as Sequencias
3 - Imprimir Intervalo das Sequencias
4 - Imprimir Elemento das Sequencias
5 - Sair
Opcao:
```

Figura2: Teste-P2.1

O problema 2.1 propõe a implementação de um programa que gere sequências conhecidas, como Fibonacci, Lucas, Pell, etc, usando herança e polimorfismo. Depois disso foi pedido que criássemos métodos que pudessem manipular essas sequências e um container para armazená-las. Optamos pela implementação de um método virtual puro na classe abstrata que retorna o ponteiro para a sequência derivada. Assim, pudemos implementar métodos genéricos, como por exemplo, `length()`, `print()`, `elem()`, etc, que funcionam para qualquer tipo de sequência,. Em relação às sequências geradas, houve uma divergência de informações entre sites em português e inglês, pois alguns consideravam “0” como início de uma determinada sequência, outros não, então optamos por seguir as sequências determinadas no roteiro do trabalho, implementando algumas exceções no momento de geração da sequência. A grande desvantagem dessa implementação, na qual acessamos os atributos da classe derivada a partir da referência para a sequência, é a perda dos métodos constantes. Uma vez que podemos modificar uma variável a partir de seu ponteiro, o compilador não permite que nossos métodos que recebem esse ponteiro seja `const`, pois ele não pode impedir essa modificação. Por outro lado, o entendimento de herança e polimorfismo se torna bem mais claro. Os comentários desse problema foram feitos apenas nos arquivos `main.cpp`, `Seq.cpp`, `Container.cpp`, `Fibonacci.cpp`, `Seq.h`, `Container.h`, `Fibonacci.h`, já os códigos se repetem bastante.

2.3.2 – PROBLEMA 2.2



```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP4\2.2\bin\Debug\2.exe
Elementos da Matriz A:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Maior valor na Matriz A: 25

Matriz A:
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

Matriz B:
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

Matriz C:
4 10 18
6 12 16

D=A+B
2 4 6 8 10
12 14 16 18 20
22 24 26 28 30
32 34 36 38 40
42 44 46 48 50

D=A+C
Tratamento de Erro
Tamanho de matrizes diferentes, erro na linha 358.

D+=B
3 6 9 12 15
18 21 24 27 30
33 36 39 42 45
48 51 54 57 60
63 66 69 72 75

D=A-B
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

D-=B
-1 -2 -3 -4 -5
-6 -7 -8 -9 -10
-11 -12 -13 -14 -15
-16 -17 -18 -19 -20
-21 -22 -23 -24 -25

D=A*B
Tratamento de Erro
Matrizes incompatíveis, erro na linha 423.
```

Figura3: Teste-P2.2

No Problema 2.2 foi proposta a criação de um template para a classe Matriz, com o objetivo de que seja possível criar matrizes de qualquer tipo de dado. Toda a implementação foi feita no arquivo .h, pois o template não é compilado apenas uma vez, mas várias vezes, para cada tipo de dado, e a única maneira portátil que permite isso é fazer toda a implementação no cabeçalho. Outra implementação imposta para o problema foi a criação de um iterator para a matriz. Criamos uma classe aninhada para essa funcionalidade e sobrecarregamos todos os operadores necessários para

manipulação. Devido à alguns problemas técnicos não implementamos a sobrecarga para o operador--, uma vez que o número das linhas e das colunas da matriz afetavam o acesso da mesma. Quando tentávamos começar do número de linhas - 1, o loop do decremento não chegava em 0. Como o tempo estava curto, preferimos deixar sem essa funcionalidade. Outra implementação para a classe matriz foi a de lançamento de exceções, usando o throw, try e catch. Criamos uma classe para cada exceção, 3 no total: matrizes de tamanhos diferentes, incompatibilidade de matrizes para a multiplicação e tratamento de acesso. Como criamos uma classe específica para cada exceção, optamos por criar um método msg() que retorna a mensagem de erro com a linha onde o throw foi executado. Por último, foi pedido que criássemos uma função global que recebesse 2 iterators quaisquer e retornasse o maior. O problema com essa função é que é necessário que o iterator tenha o operador > sobrecarregado. No caso do iterator matriz funcionou, pois comparamos o valor diretamente, mas não funcionou para outros tipos de iterator como o vector, por conta da comparação *if((be++) > *maximo)*. Foi necessário fazer a comparação direta com o be++, pois se eu incrementar o be, depois comparar ou até mesmo imprimir, por algum motivo desconhecido ele imprime apenas alguns valores da matriz, e não todos, por isso essa função não funcionou totalmente. No final das contas faltou muita coisa para que essa questão estivesse realmente pronta.

2.3.3 – PROBLEMA 2.3

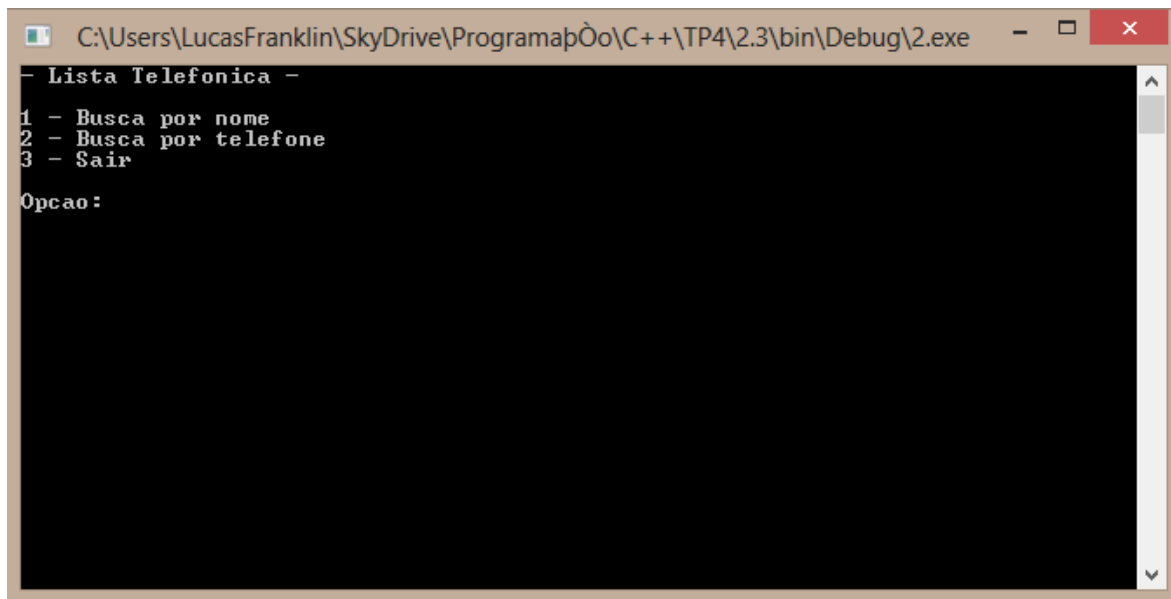
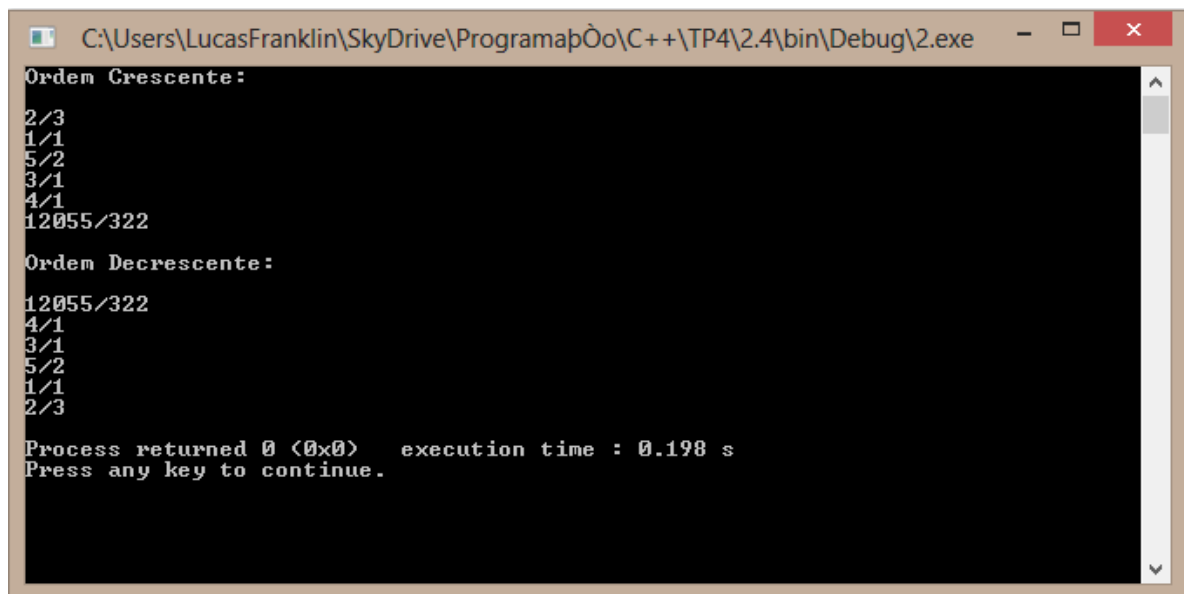


Figura4: Teste-P2.3

No problema 2.3 foi proposto que criássemos uma agenda telefônica usando map e multimap. Maps e Multimaps são containers associativos que armazenam elementos formados pela combinação de uma chave e um valor mapeado. A diferença entre eles é que o multimap suporta múltiplos elementos com a mesma chave. Infelizmente o programa só busca por nomes exatos.

2.3.4 – PROBLEMA 2.4



```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP4\2.4\bin\Debug\2.exe
Ordem Crescente:
2/3
1/1
5/2
3/1
4/1
12055/322
Ordem Decrescente:
12055/322
4/1
3/1
5/2
1/1
2/3
Process returned 0 (0x0)   execution time : 0.198 s
Press any key to continue.
```

Figura5: Teste-P2.4

O problema 2.4 propôs algumas modificações e implementações na classe Racional para podermos ordenar um container da mesma. Foi criada uma sobrecarga do operador `()` para comparação de 2 racionais diferentes, retornando `true` ou `false` caso um fosse ou não menor que o outro. Usando o `istream_iterator` percorremos todo o arquivo contendo valores racionais no formato `x/y`. A partir do operador de fluxo de saída `>>` pegamos esses valores que eram `string` e os convertemos em `int`, salvando-os no objeto. Depois, percorrendo todas as posições com esse iterator, criamos um container de racionais que pode ser ordenado usando a função `sort()` da biblioteca `algorithm`. Usando o método `reverse()` no vetor já ordenado pudemos obter a saída em ordem inversa. Para a impressão na tela, usamos o `ostream_iterator`. Esses 2 iterators facilitam bastante a vida do desenvolvedor quando se deseja imprimir containers.

2.3.5 – PROBLEMA 2.5

Pelo fato do grande número de problemas que ocorreram na implementação das questões anteriores, principalmente no problema 2.2, infelizmente não houve tempo hábil para o desenvolvimento dessa questão.

3 - CONCLUSÃO

Sem dúvida esse foi o trabalho mais difícil de todos, porém o mais recompensador e o que mais nos agregou conhecimento. Por questão de tempo não conseguimos nem iniciar o problema 2.5. Ficamos bastante tempo na questão 2.2, inicialmente tentando entender como funcionava iterator e depois lidando com as centenas de erros e problemas que apareceram durante a implementação. No geral, esse último trabalho não ficou muito bom. Além da última questão que deixamos de fazer, todas as outras deixaram um pouco a desejar, pelo menos no nosso ponto de vista. Acreditamos que não conseguimos gerenciar nosso tempo como um todo, para dar conta das outras disciplinas e desse trabalho. No problema 2.2, muitos problemas continuam ocorrendo, além de alguns métodos básicos que ficaram a ser implementados, contudo, ele funciona. Apesar de todas as ocorrências, valeu a pena todo o trabalho.

4 – BIBLIOGRAFIA

Cplusplus.com. [Online] [Citado em: 06 de 12 de 2013.] <http://www.cplusplus.com/>.

Stack Overflow. [Online] [Citado em: 06 de 12 de 2013.] <http://stackoverflow.com>.

Yolinux. [Online] [Citado em: 06 de 12 de 2013.]

<http://www.yolinux.com/TUTORIALS/CppStlMultiMap.html>

CPP Reference. [Online] [Citado em: 06 de 12 de 2013.] <http://en.cppreference.com>.