

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

Diogo Nascimento Silva
Lucas Franklin Silva

TRABALHO PRÁTICO III
PROGRAMAÇÃO ORIENTADA A OBJETOS

Montes Claros - MG
Novembro / 2013

1 – INTRODUÇÃO

Esse trabalho teve como objetivo a aplicação de conceitos de orientação a objetos aprendidos em aula usando a linguagem C++. Nesse trabalho prático usamos as implementações do trabalho anterior, adicionando construtores de cópia, construtores de conversão, namespaces e sobrecarga de operadores. Alguns padrões foram adotados por nós para que a codificação ficasse organizada e legível. O código foi indentado usando o plugin Source Code Formater (AStyle) presente no Code::Blocks.

2 – DESENVOLVIMENTO

2.1 - PADRÕES

Para o desenvolvimento dos códigos foram adotados os seguintes padrões:

- Nomes de variáveis, classes e métodos em inglês, com exceção das já especificadas no trabalho.
- Nomes de classes sempre iniciam com letra maiúscula.
- Padrão lowerCamelCase, variação do CamelCase, para nomes de métodos e variáveis.
- Prefixo get para métodos com a finalidade de retornar atributos de uma determinada classe.
- Prefixo set para métodos com a finalidade de setar atributos de uma determinada classe.
- Prefixo print para métodos com a finalidade de imprimir informações do objeto
- Funções que não modificam o objeto são constantes.
- Retorno dos operadores constantes, para que a manipulação indevida dos resultados seja impedida.
- Funções que são simples e pequenas são inline.

2.2 – COMENTÁRIOS

A maior parte do código, mesmo as mais óbvias e simples, foram comentadas para melhor entendimento.

2.3 – TESTES E IMPLEMENTAÇÃO

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

2.3.1 – PROBLEMA 2.1

```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP3\2.1\bin\Debug\2.exe
CVetor:
Elementos A: 1 2 3
Elementos B: 2 4 6
Soma A+B: 3 6 9
Subtracao A-B: -1 -2 -3
Produto Escalar A*B: 28
Produto Vetorial AxB: 0 0 0
Divisao B/A: 2 2 2

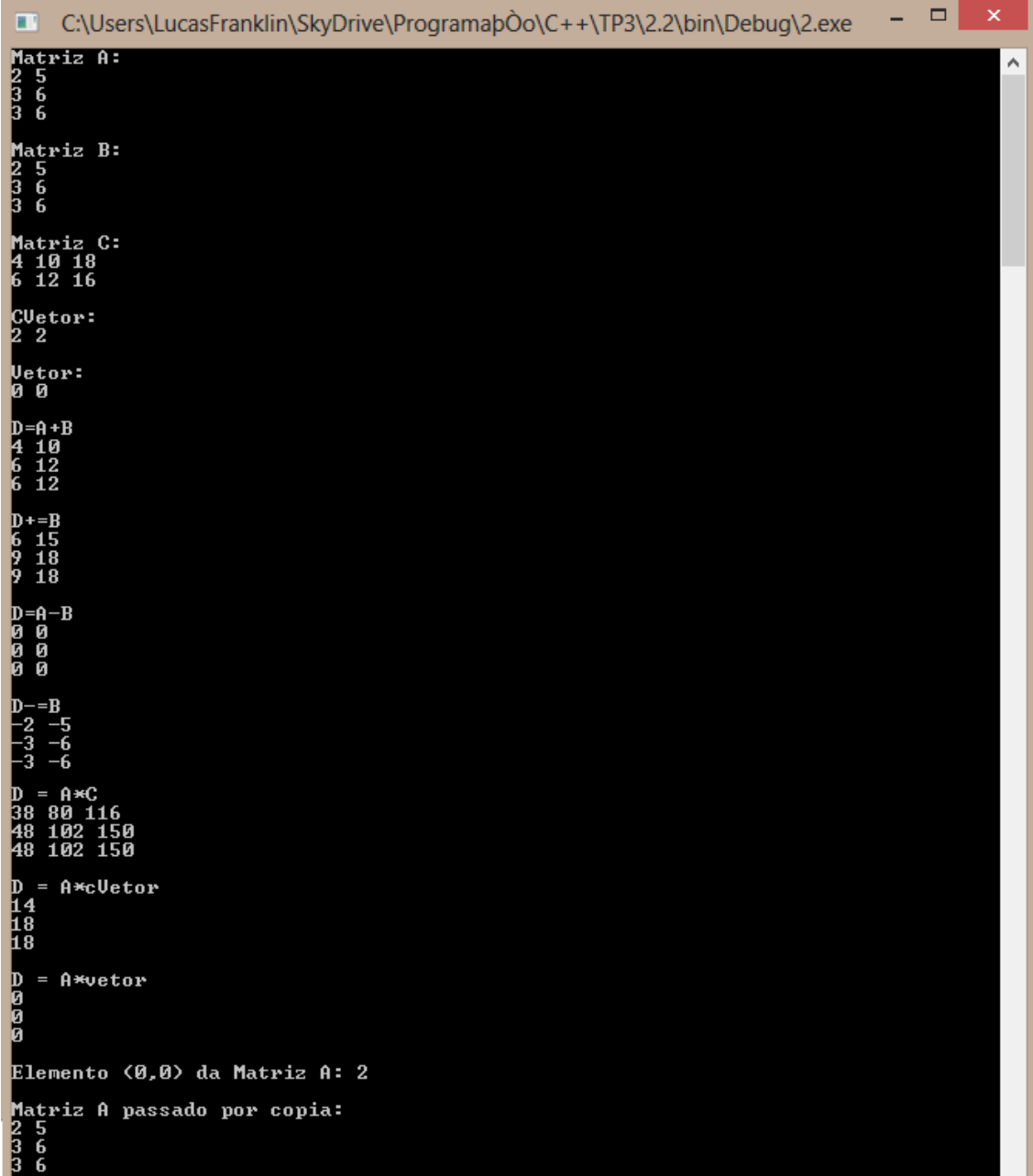
Process returned 0 (0x0)   execution time : 0.097 s
Press any key to continue.
```

Figura1: Teste-P2.1

O problema 2.1 propõe a implementação de uma classe CVetor, que é basicamente um vetor, com atributos inteiros, que pode ser alocado dinamicamente dependendo da necessidade do usuário. Ele pertence ao namespace DataStructures, todos os operadores básicos foram sobrecarregados para uma determinada operação. A maioria dos operadores tem retorno constante por cópia. O retorno é por cópia para garantir a consistência dos dados uma vez que o objeto instanciado temporariamente dentro do operador sobrecarregado é destruído ao final da operação, e é constante para impedir que esse objeto seja modificado antes que a operação seja concluída. O operador % e %= é usado para multiplicação escalar, e é permitido apenas para operações entre vetores de dimensão três, por razões técnicas. O operador * e * realiza

multiplicação escalar, retornando um resultado inteiro. Os demais operadores realizam sua determinada operação.

2.3.2 – PROBLEMA 2.2



```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP3\2.2\bin\Debug\2.exe
Matriz A:
2 5
3 6
3 6

Matriz B:
2 5
3 6
3 6

Matriz C:
4 10 18
6 12 16

CVetor:
2 2

Vetor:
0 0

D=A+B
4 10
6 12
6 12

D+=B
6 15
9 18
9 18

D=A-B
0 0
0 0
0 0

D-=B
-2 -5
-3 -6
-3 -6

D = A*C
38 80 116
48 102 150
48 102 150

D = A*cVetor
14
18
18

D = A*vetor
0
0
0

Elemento <0,0> da Matriz A: 2

Matriz A passado por copia:
2 5
3 6
3 6
```

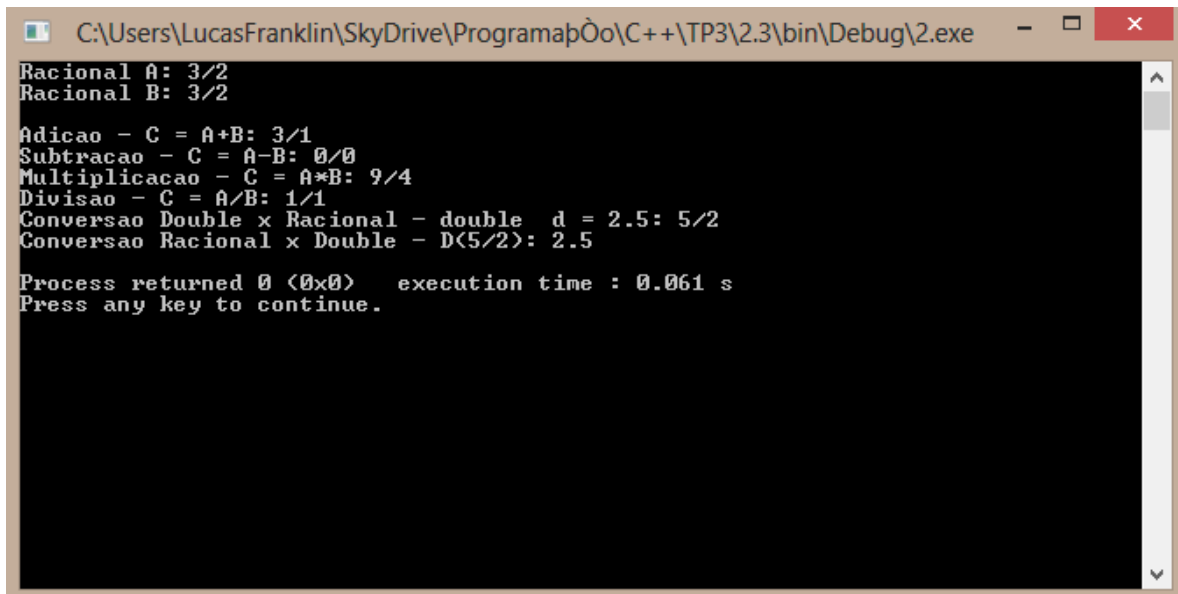
Figura2: Teste-P2.2

No Problema 2.2 foi proposto que mudássemos a classe Matriz do trabalho anterior, criando sobrecarga de operadores, construtores de cópia, etc. Foi criado construtor de cópia e sobrecarga do operador de atribuição para a cópia da matriz.

Classes com atributos dinâmicos necessitam de um construtor de cópia e operador de atribuição personalizado, pois podem ocorrer diversos tipos de erro ao tentar copiar um atributo dinâmico. No Trabalho Prático 2 não sentimos falta desses recursos porque não fizemos nenhuma cópia da matriz. A maioria dos operadores foram sobrecarregados, e realizam sua respectiva operação, damos destaque para o de multiplicação, que pode ser utilizado para multiplicar duas matrizes, um vetor x matriz ou CVetor x matriz. Um sobrecarga do método printMatrix() foi criada apenas para mostrar que o construtor de cópia funciona corretamente. A classe foi adicionada ao namespace Matematica.

Correção em relação ao Trabalho Prático 2: Trocamos o nome do método getMatrix(), que imprimia a matriz, por printMatrix() para obedecer aos padrões que foram adotados.

2.3.3 – PROBLEMA 2.3

A screenshot of a Windows command prompt window titled "C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP3\2.3\bin\Debug\2.exe". The window has a black background with white text. The text shows the execution of a C++ program that performs various operations on rational numbers. The operations include addition, subtraction, multiplication, division, and conversion between rational numbers and doubles. The program ends with a message "Process returned 0 (0x0) execution time : 0.061 s" and a prompt "Press any key to continue.".

```
Racional A: 3/2
Racional B: 3/2

Adicao - C = A+B: 3/1
Subtracao - C = A-B: 0/0
Multiplicacao - C = A*B: 9/4
Divisao - C = A/B: 1/1
Conversao Double x Racional - double d = 2.5: 5/2
Conversao Racional x Double - D(5/2): 2.5

Process returned 0 (0x0) execution time : 0.061 s
Press any key to continue.
```

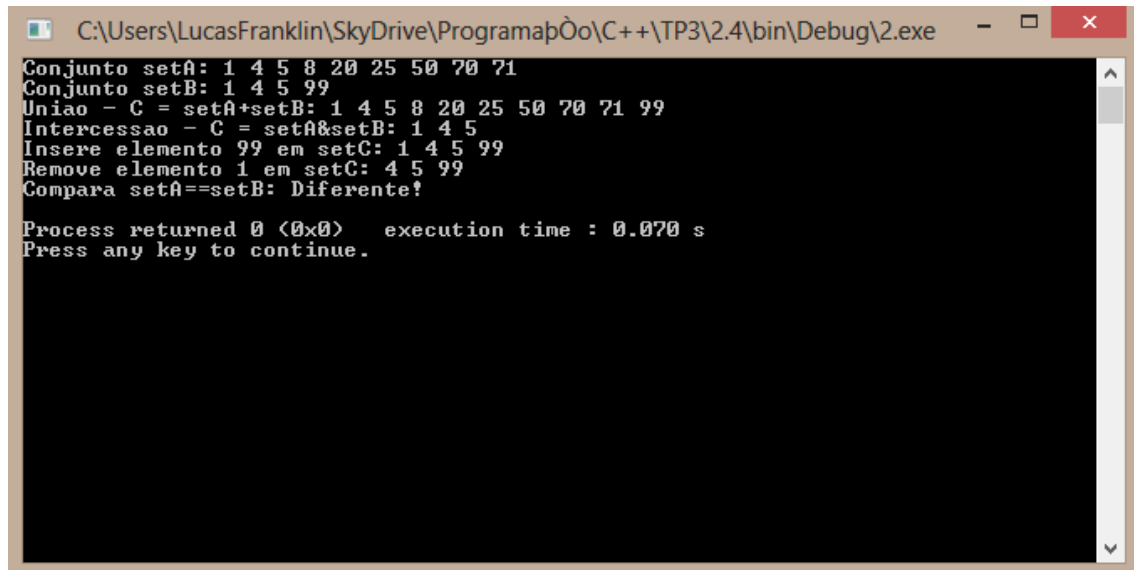
Figura3: Teste-P2.3

No problema 2.3 foi proposto que mudássemos a classe Racional do trabalho anterior, adicionando sobrecarga de operadores, construtores de cópia, conversão de tipos e o namespace Matematica. Os operadores sobrecarregados realizam sua determinada operação.

Usamos a função modf() da biblioteca <cmath>, essa função retorna a parte fracionária de um valor do tipo double, assim conseguimos converter um Double em uma fração utilizando um método que consta na referencia desse trabalho.

Correção em relação ao Trabalho Prático 2: Trocamos os nomes dos métodos `getDenominatorErrorMessage()` e o `getRacional()`, que imprimiam informações na tela, para `printDenominatorErrorMessage()` e `printRacional()` para se encaixarem nos padrões estabelecidos.

2.3.4 – PROBLEMA 2.4



```
C:\Users\LucasFranklin\SkyDrive\Programas\OoC++\TP3\2.4\bin\Debug\2.exe
Conjunto setA: 1 4 5 8 20 25 50 70 71
Conjunto setB: 1 4 5 99
Uniao - C = setA+setB: 1 4 5 8 20 25 50 70 71 99
Intersecao - C = setA&setB: 1 4 5
Insere elemento 99 em setC: 1 4 5 99
Remove elemento 1 em setC: 4 5 99
Compara setA==setB: Diferente!

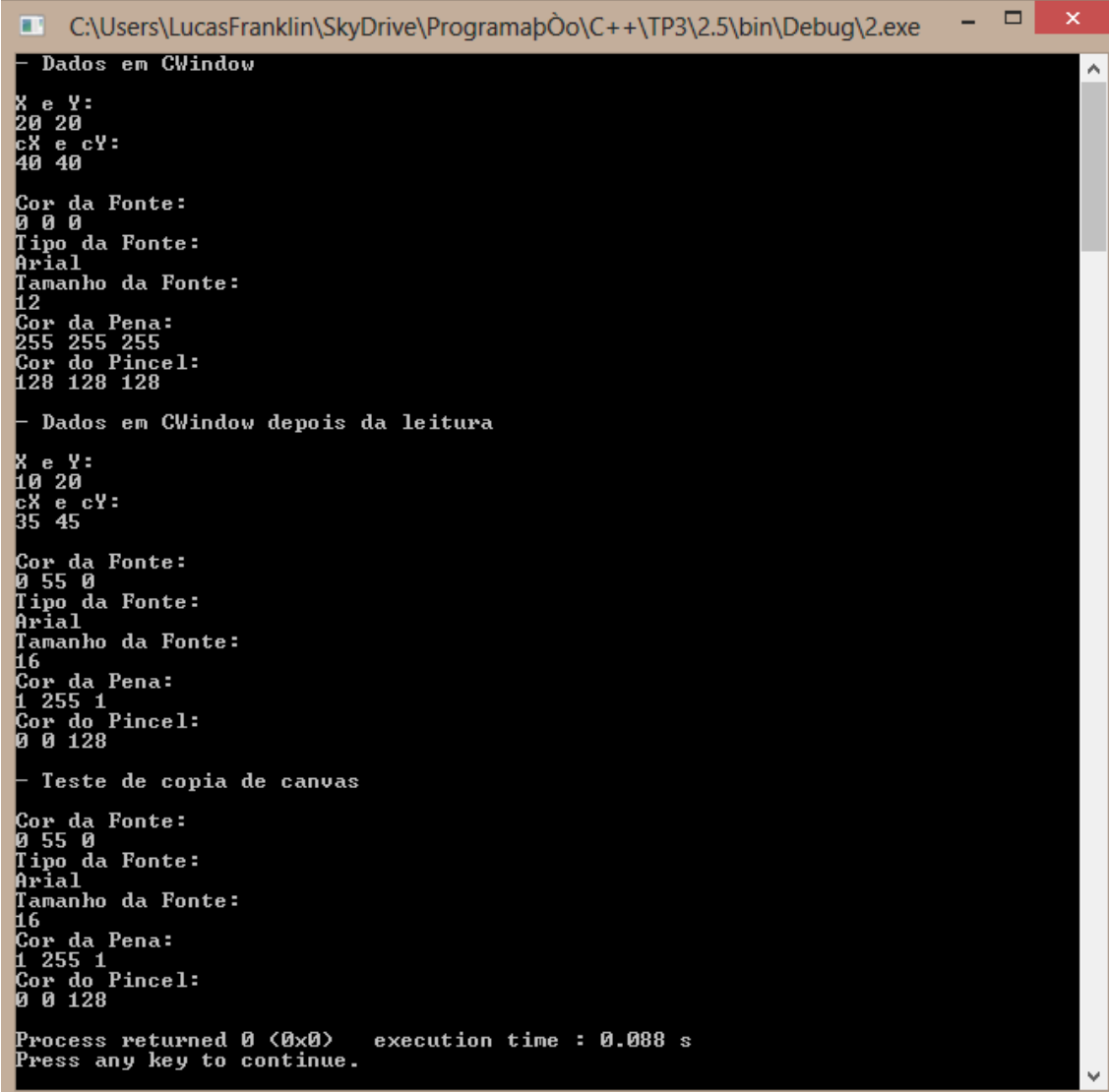
Process returned 0 (0x0)   execution time : 0.070 s
Press any key to continue.
```

Figura4: Teste-P2.4

O problema 2.4 propôs modificarmos a classe `IntegerSet` do trabalho anterior, adicionando sobrecarga de operadores e o namespace `Matematica`. A sobrecarga dos operadores apenas invocam os métodos já existentes na classe que foram desenvolvidos no trabalho anterior. Os operadores sobrecarregados realizam as operações que foram determinadas no roteiro desse trabalho.

Correção em relação ao Trabalho Prático 2: Trocamos o nome do método `getElements()`, que imprimia o conjunto na tela, para `printElements()`, ficando assim de acordo com os padrões adotados.

2.3.5 – PROBLEMA 2.5



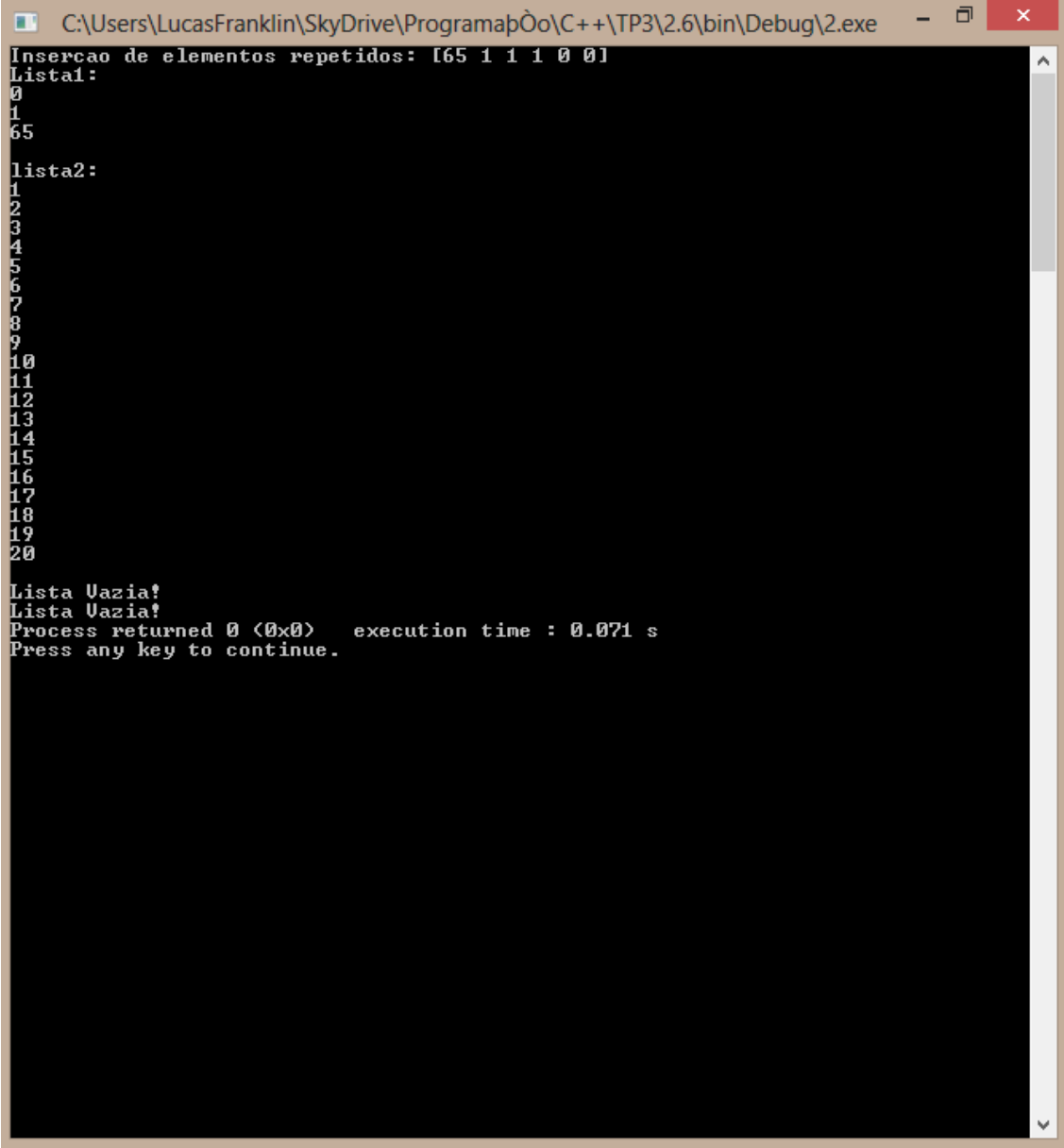
```
C:\Users\LucasFranklin\SkyDrive\Programapão\C++\TP3\2.5\bin\Debug\2.exe
- Dados em CWindow
X e Y:
20 20
cX e cY:
40 40
Cor da Fonte:
0 0 0
Tipo da Fonte:
Arial
Tamanho da Fonte:
12
Cor da Pena:
255 255 255
Cor do Pincel:
128 128 128
- Dados em CWindow depois da leitura
X e Y:
10 20
cX e cY:
35 45
Cor da Fonte:
0 55 0
Tipo da Fonte:
Arial
Tamanho da Fonte:
16
Cor da Pena:
1 255 1
Cor do Pincel:
0 0 128
- Teste de copia de canvas
Cor da Fonte:
0 55 0
Tipo da Fonte:
Arial
Tamanho da Fonte:
16
Cor da Pena:
1 255 1
Cor do Pincel:
0 0 128
Process returned 0 (0x0) execution time : 0.088 s
Press any key to continue.
```

Figura5: Teste-P2.5

No problema 2.5 foi proposto que modificássemos as classes CWindow e CWindowSingleton, adicionando a sobrecarga de operadores de fluxo para entrada e saída, criar um construtor de cópia e o operador de atribuição para Canvas. Foram criados construtores de cópia e operadores de atribuição para todas as Classes desse problema. Para o CWindowSingleton colocamos o construtor de cópia e o operador de atribuição privados, para que a cópia do mesmo seja impedida, assegurando apenas uma instancia para o Singleton.

Correção em relação ao Trabalho Prático 2: Correção do nome da classe CWindowingleton para CWindowSingleton.

2.3.6 – PROBLEMA 2.6



```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP3\2.6\bin\Debug\2.exe
Insercao de elementos repetidos: [65 1 1 1 0 0]
Lista1:
0
1
65

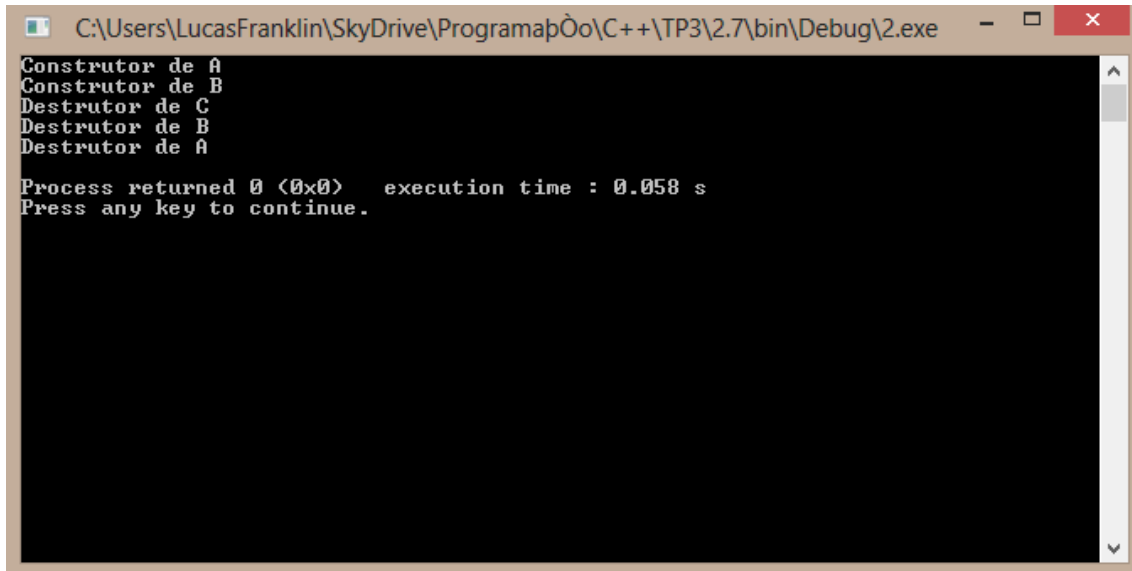
lista2:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Lista Vazia!
Lista Vazia!
Process returned 0 (0x0)   execution time : 0.071 s
Press any key to continue.
```

No problema 2.6 foi proposto que criássemos construtor de cópia e sobrecarga de operadores para a classe List do trabalho anterior. Inicialmente por problemas com o ponteiro Next não foi possível concluir o desenvolvimento do construtor de cópia e a sobrecarga do operador de atribuição, como reservamos pouco tempo para essa questão o tempo não foi suficiente para a solução desse problema. Foi desenvolvida apenas a sobrecarga para os operadores de inserção(+=), remoção(-=), fluxo(<< e >>) e impressão(). Os operadores de concatenação ou não funcionaram, ou dependiam do operador de atribuição.

Correção em relação ao Trabalho Prático 2: Correção do nome da classe Stack, que é para pilha, para List, que é para lista.

2.3.7 – PROBLEMA 2.7



```
C:\Users\LucasFranklin\SkyDrive\Programas\Oo\C++\TP3\2.7\bin\Debug\2.exe
Construtor de A
Construtor de B
Destrutor de C
Destrutor de B
Destrutor de A
Process returned 0 (0x0) execution time : 0.058 s
Press any key to continue.
```

Figura7: Teste-P2.7

O problema 2.7 introduz conceitos de classes derivadas e herança. Pelo que podemos perceber a classe derivada C utiliza o construtor de A, no momento que é instanciada, ao final do programa a instancia de C é destruída, depois a de B, que é um atributo de C e por último A, que é a classe pai.

3 - CONCLUSÃO

Nesse trabalho prático III implementamos sobrecargas de operadores, construtores de cópia, conversão de tipos e namespaces. Entre todos os novos elementos adicionados o namespace foi o mais difícil de encontrar boas referências. Implementar as sobrecargas para os operadores foi bastante simples, uma vez que a maior parte da codificação do mesmo já havia sido implementada no trabalho anterior. A nosso ver a coisa mais importante aprendida nesse trabalho foi a importância da criação dos construtores de cópia e sobrecarga do operador de atribuição para a cópia de objetos com atributos dinâmicos.

4 – BIBLIOGRAFIA

Cplusplus.com. [Online] [Citado em: 02 de 11 de 2013.] <http://www.cplusplus.com/>.

Stack Overflow. [Online] [Citado em: 02 de 11 de 2013.] <http://stackoverflow.com>.

Só Matemática. [Online] [Citado em: 30 de 09 de 2013.]

<http://www.somatematica.com.br/fundam/decimais/decimais4.php>.