**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

**CAMPUS SÃO CARLOS**

**JOÃO VITOR BRANDÃO MOREIRA - 407496**

**LUCAS OLIVEIRA DAVID – 407917**

**THIAGO FARIA NOGUEIRA - 407534**

**DOCUMENTAÇÃO TRABALHO 2 - ESTRUTURAS DE DADOS**

SÃO CARLOS – SP

2012

**1. INTRODUÇÃO**

Este trabalho consiste na criação e teste das seguintes estruturas de dados estudadas ao longo do curso: fila, lista duplamente encadeada com header, cadastro e árvore binária de busca. Desde o início, definimos como objetivo a implementação por meio de Orientação a Objetos, o que nos direcionou para a linguagem de programação C++. É interessante frisar, desde já, que todas as estruturas anteriores possuem um elemento em comum: o nó. Além disse, todas as classes foram implementadas utilizando *template* afim de criar estruturas de dados genéricas.

**2. ESTRATÉGIA DE IMPLEMENTAÇÃO**

Criamos primeiramente a classe *Node*, isto é, Nó, que tem os seguintes atributos: **T item; Node \*esq; Node \*dir**, e definimos operações de construção (com e sem parâmetro) e destrução. Tal classe seria, a partir daí, utilizada por todas as outras implementações (listas e árvores, por exemplo).

Fizemos a lista duplamente encadeada com header, também tendo em mente a reutilização de código. Essa estrutura não tem regras para inserção ou remoção, portanto fizemos métodos de inserção e remoção genéricos, além de métodos para calcular o tamanho da estrutura, imprimir e fazer busca.

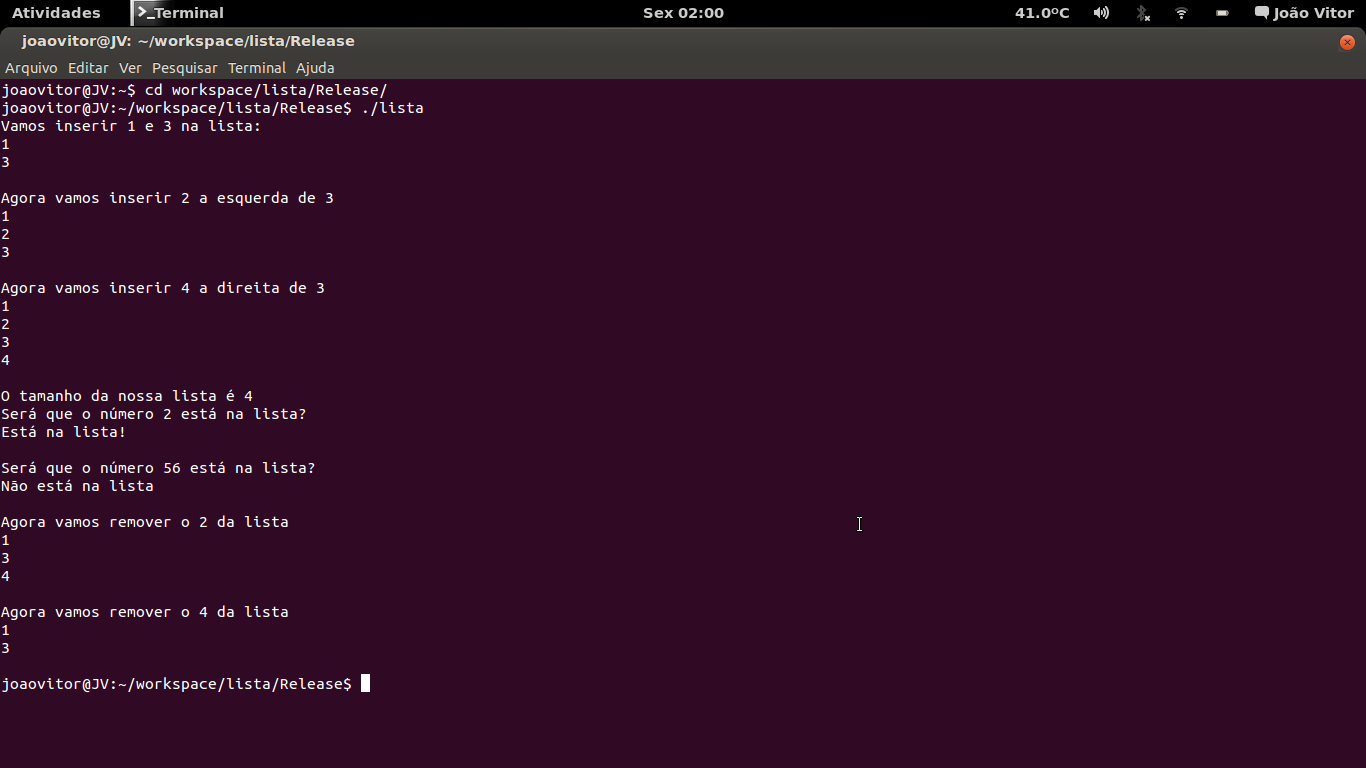
As classes Cadastro e Lista herdam de forma privada da classe Lista. Mas porque? Justamente para poder utilizar os métodos da Lista para implementar as operações inerentes à cada classe sem que os métodos da Lista fiquem acessíveis fora destas classes. Em outras palavras, utilizamos métodos da Lista para construir métodos do Cadastro e da Fila, mas os métodos da Lista não podem ser chamados fora do escopo destas classes. Assim o conceito de TAD não é quebrado.

A classe *Tree* (que é, essencialmente, uma árvore binária de busca), por sua vez, possui um único atributo: **Node<T> \*root**, que é sua raíz. Mais importante, este mesmo possúi os métodos básicos para o comportamento da árvore: **inserir**, **remover**, **buscar**, **percurso em ordem**, **pré-ordem** e **pós-ordem**, **busca**, **vazio?**, etc. Os metodos **remove** (que segue o padrão adotado em sala de aula, isto é, durante a remoção de um elemento com dois filhos, o maior elemento da sub-árvore esquerda é transferido para a raiz dessa mesma árvore, que é o elemento que queríamos remover inicialmente. Posteriormente, o nó do elemento transferido é removido recursivamente, evitando assim uma duplicação na árvore) e as **listagens ordenadas** foram implementados *recursivamente*.

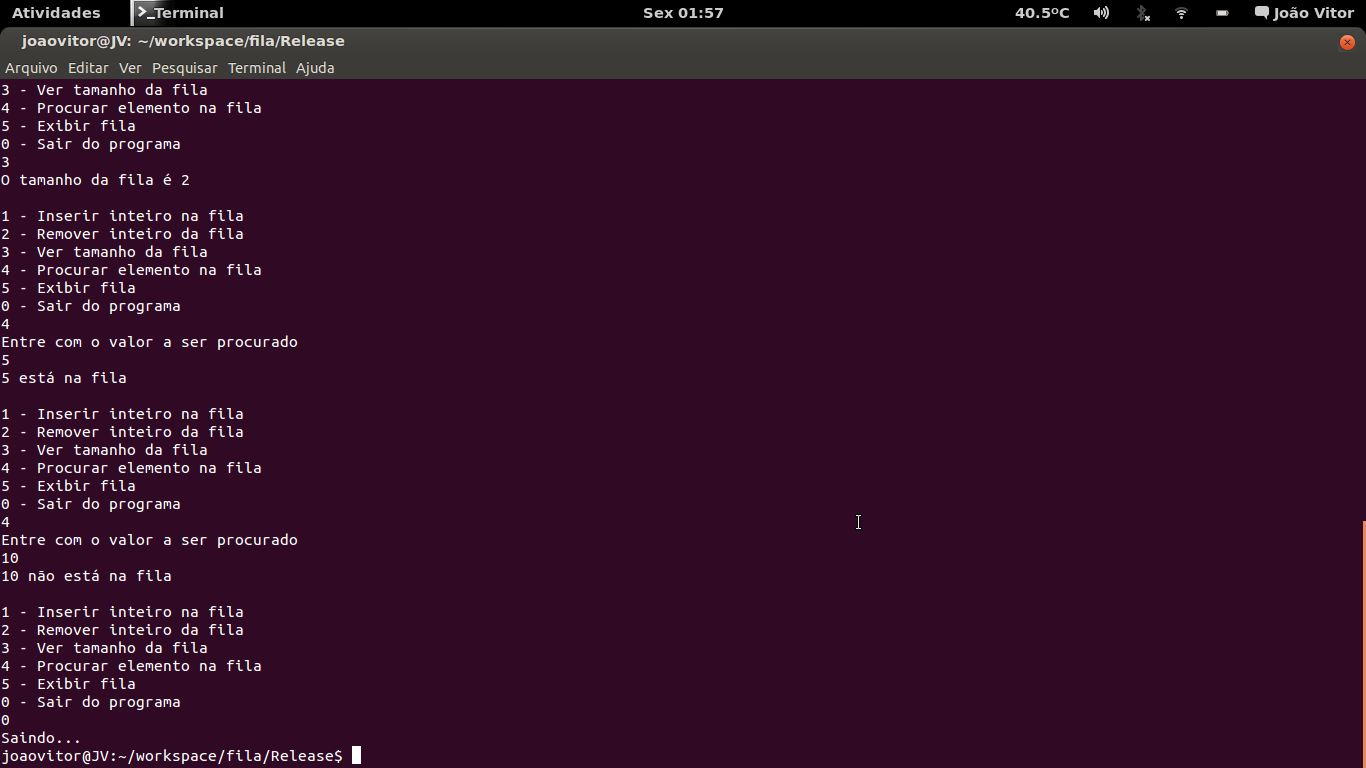
Todas as estruturas foram construídas com a utilização de *template*, a fim de proporcionar uma generalização do uso das estruturas.

**3. PRINT-SCREENS**

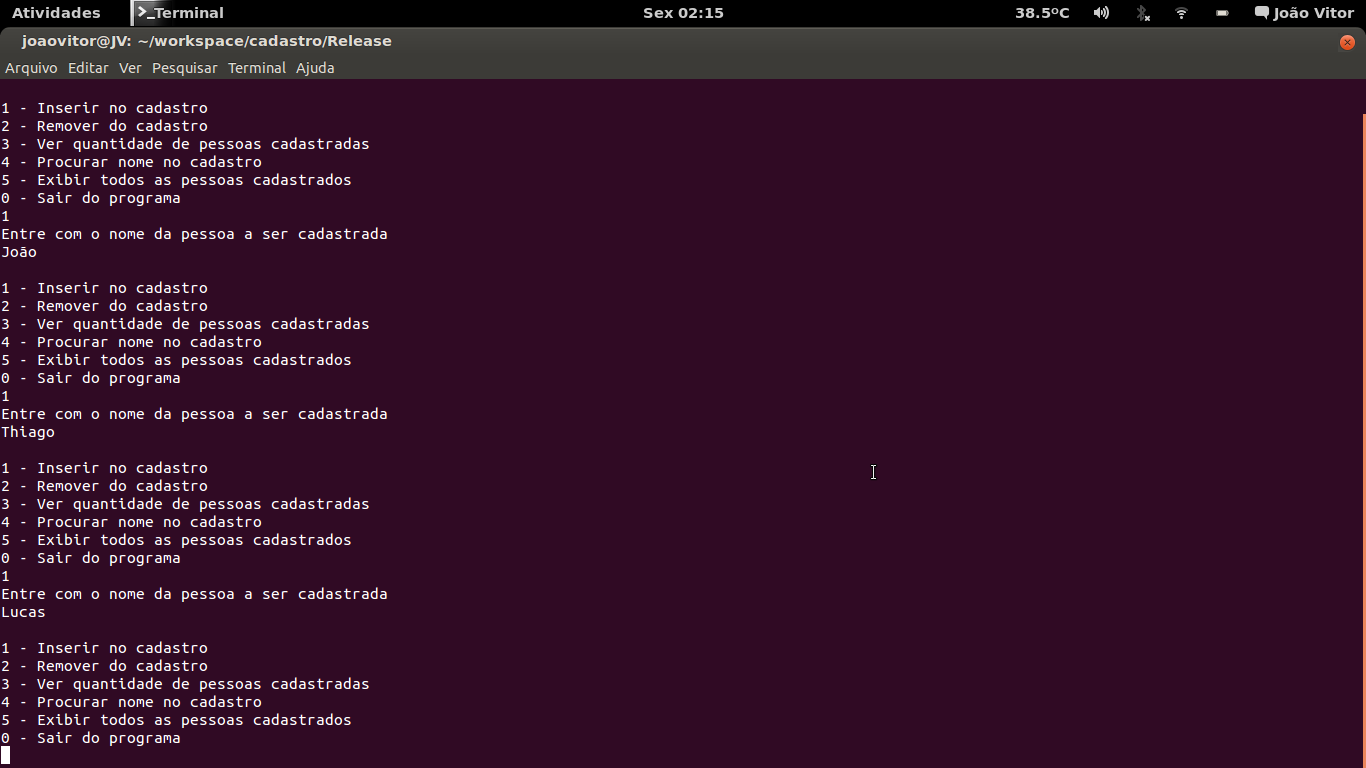
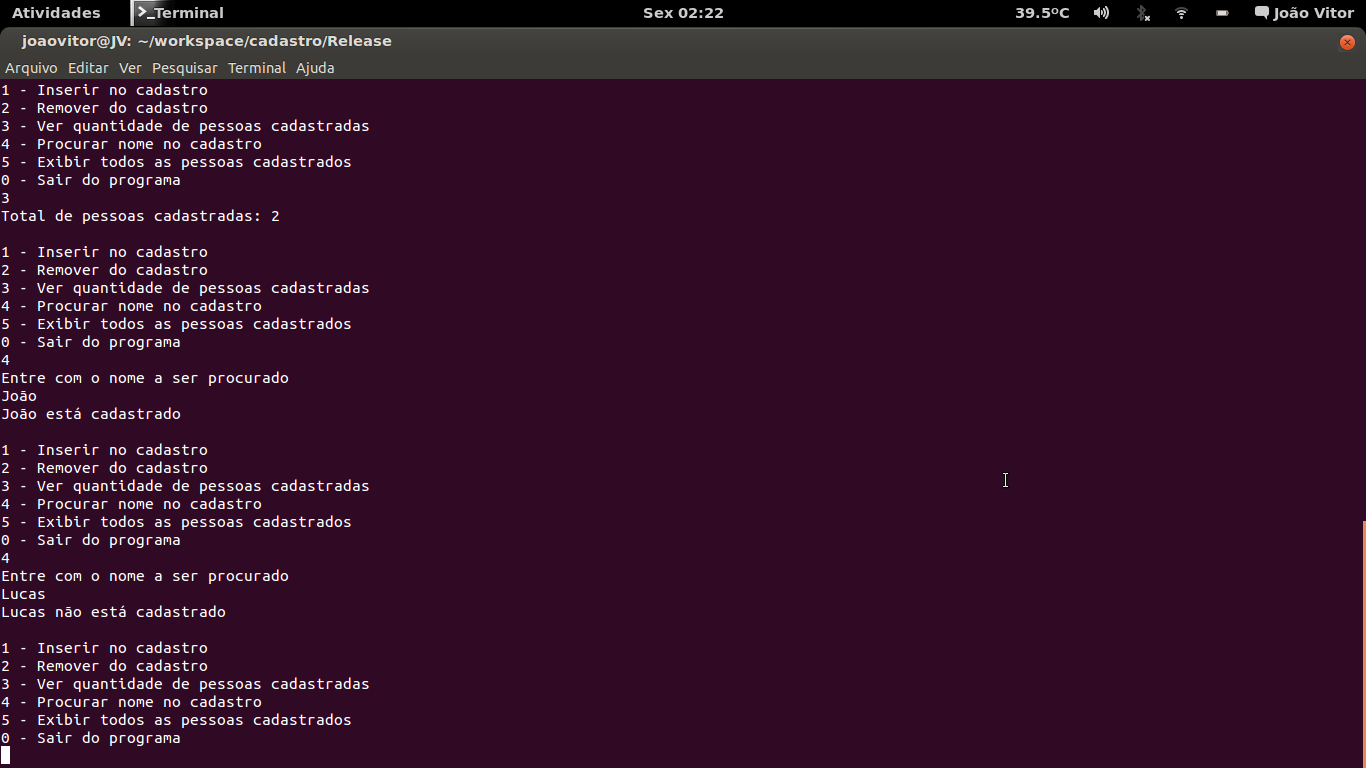
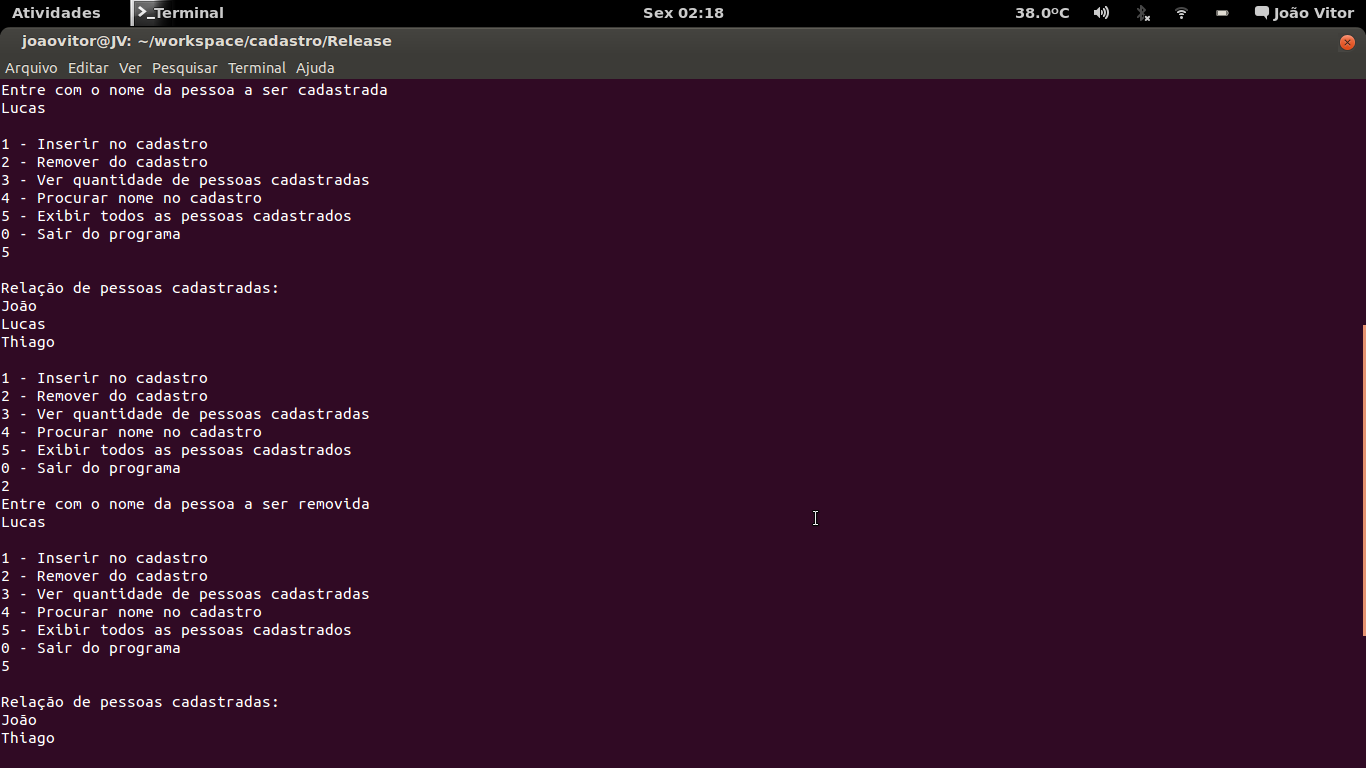
**3.1. Lista**



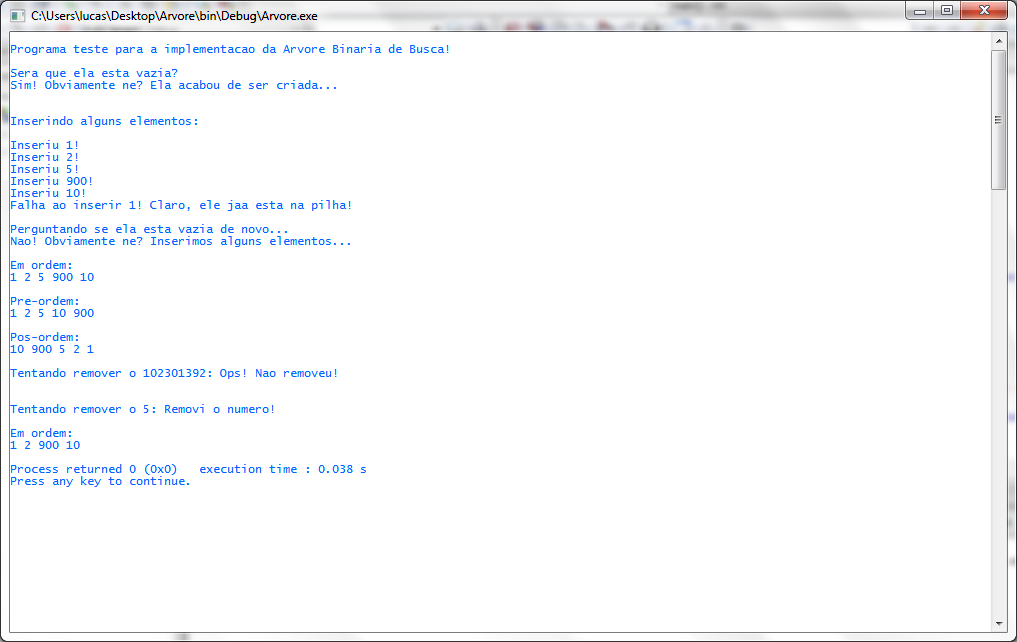
**3.2. Fila**

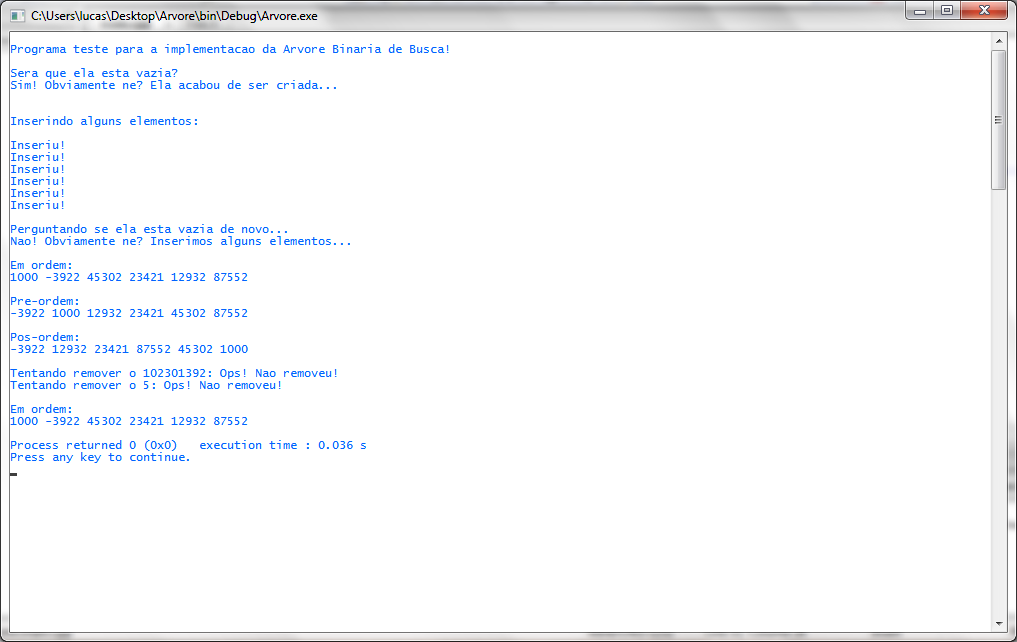
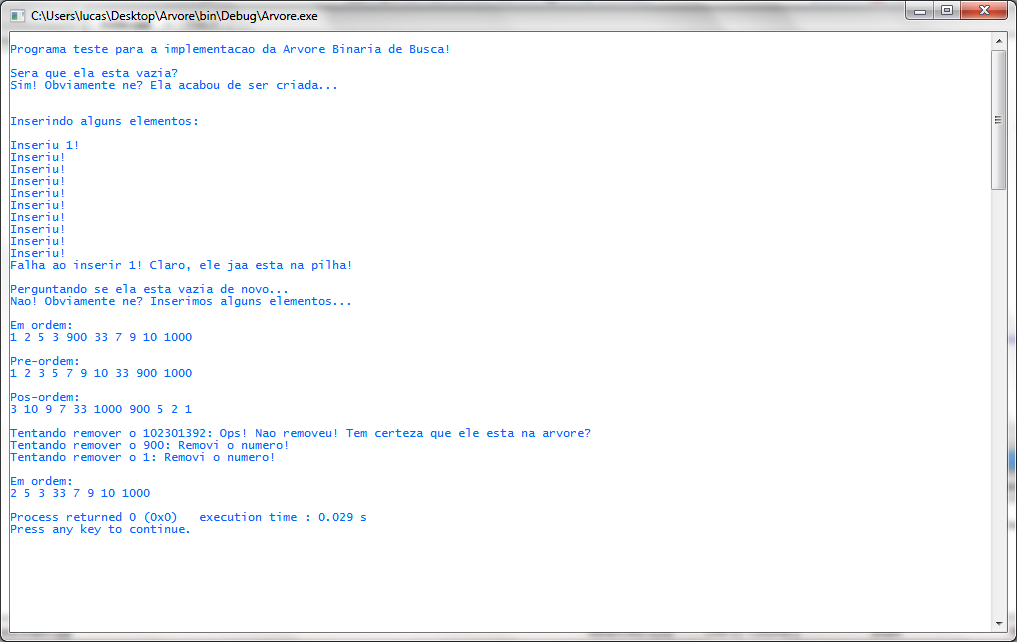
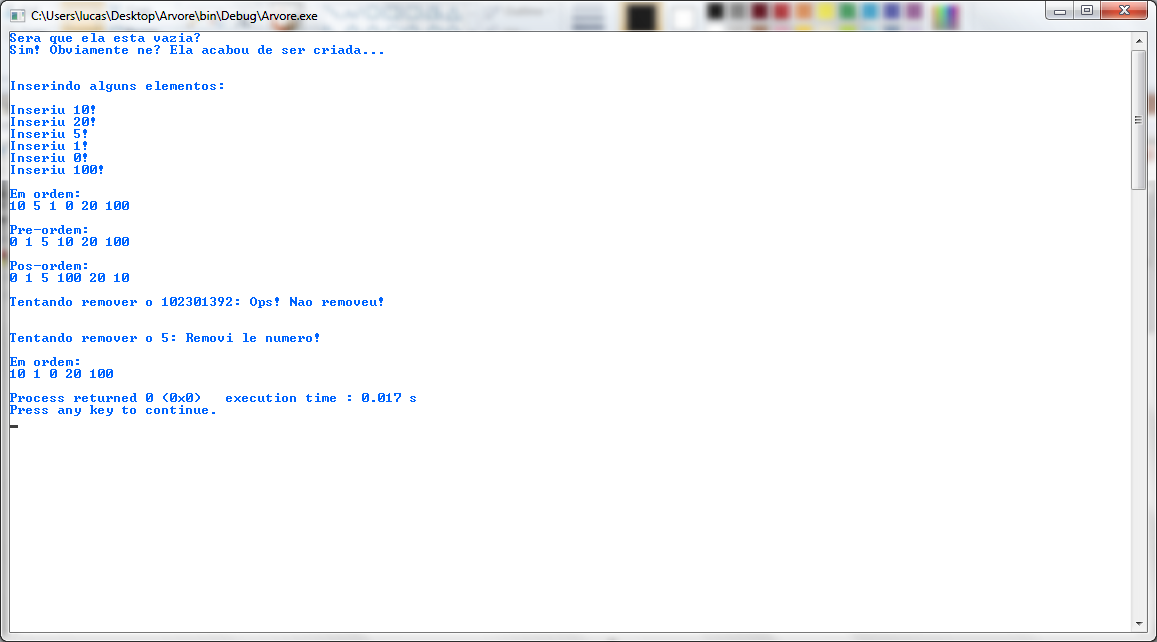


**3.3. Cadastro**

**3.4. Árvore**



**4. CONCLUSÃO**

Observamos as vantagens da implementação de estruturas genéricas, uma vez que foi definido a utilização da **lista duplamente encadeada com header** para a implementação do trabalho 3. Também ficou evidente as semelhanças e diferenças entre as estruturas utilizadas: o **Cadastro** e **Fila** são classes criadas a partir de uma *herança* da classe **Lista**, embora estes mesmos mostrem conceitos distintos de manipulação. Tal herança exprime o conceito de “reaproveitamento de código” de nosso trabalho.