Universidade Federal de São Carlos
Departamento de Computação
Bacharelado em Ciências da Computação

Estrutura de Dados
Trabalho 2:
Implementação de Estruturas Indicadas

alunos:
Cleber Takahito Kawamorita, 379425
Felipe Augusto de Salles Pupo, 379514
João Celso Santos de Oliveira, 379247
Willian Lira Cardoso, 379530

professor:
Dr. Roberto Ferrari

- São Carlos, SP -
Junho de 2011

Descrição do Trabalho

Abaixo a descrição fornecida pelo professor:

O Que Fazer?

Implementar e testar 4 estruturas de armazenamento temporário de conjuntos de informações

- *Uma estrutura do tipo Fila;*
- *Uma estrutura dentre as estudadas na unidade 9 (Listas Simples);*
- *Uma estrutura dentre as estudadas nas unidades 10 ou 15 do Livro Virtual (variações e generalizações de listas);*
- *Uma estrutura dentre as estudadas nas unidades 16 ou 17 ou 20 do Livro Virtual (Árvores).*

O objetivo é simplesmente demonstrar habilidade para implementar as estruturas de armazenamento estudadas durante a disciplina. A estratégia de desenvolvimento deve respeitar as recomendações referentes ao uso de Tipos Abstratos de Dados.

Como Fazer?

Escolher as quatro estruturas, implementar, e fazer um programa simples, em arquivo separado, para testar.

Divisão do trabalho e Estratégia

A divisão do trabalho foi feita em partes de acordo com a dificuldade. Foi gerada uma lista com as partes e cada integrante do grupo ia completando as tarefas em aberto.

As lista é mostrada abaixo:

1 - Estrutura mínima do programa (criar arquivos vazios)

2.0 - Relatório: criar estrutura mínima do relatório

2.1 - Relatório: introdução, estratégia e printscreen

2.2 - Relatório: diagrama

2.3 - Relatório: revisão (todo o grupo)

3 - Classe ELEMENTO

4 - Classe NO1

5 - Classe NO2

6.0 - Classe FILA: criar classe

6.1 - Classe FILA: métodos entra e sai

6.2 - Classe FILA: métodos vazia e cheia

7.0 - Classe LISTA_SIMPLES: criar classe

7.1 - Classe LISTA_SIMPLES: adicionarElemento e removerElemento

7.2 - Classe LISTA_SIMPLES: estaNaLista

8.0 - Classe LISTA_AVANCADA: criar classe

8.1 - Classe LISTA_AVANCADA: adicionar e retirar (retirar por valor de elemento)

8.2 - Classe LISTA_AVANCADA: estaNaLista e retirar (retirar por ponteiro de nó)

8.3 - Classe LISTA_AVANCADA: insereADireita e insereAEsquerda

9.0 - Classe ARVORE: criar classe

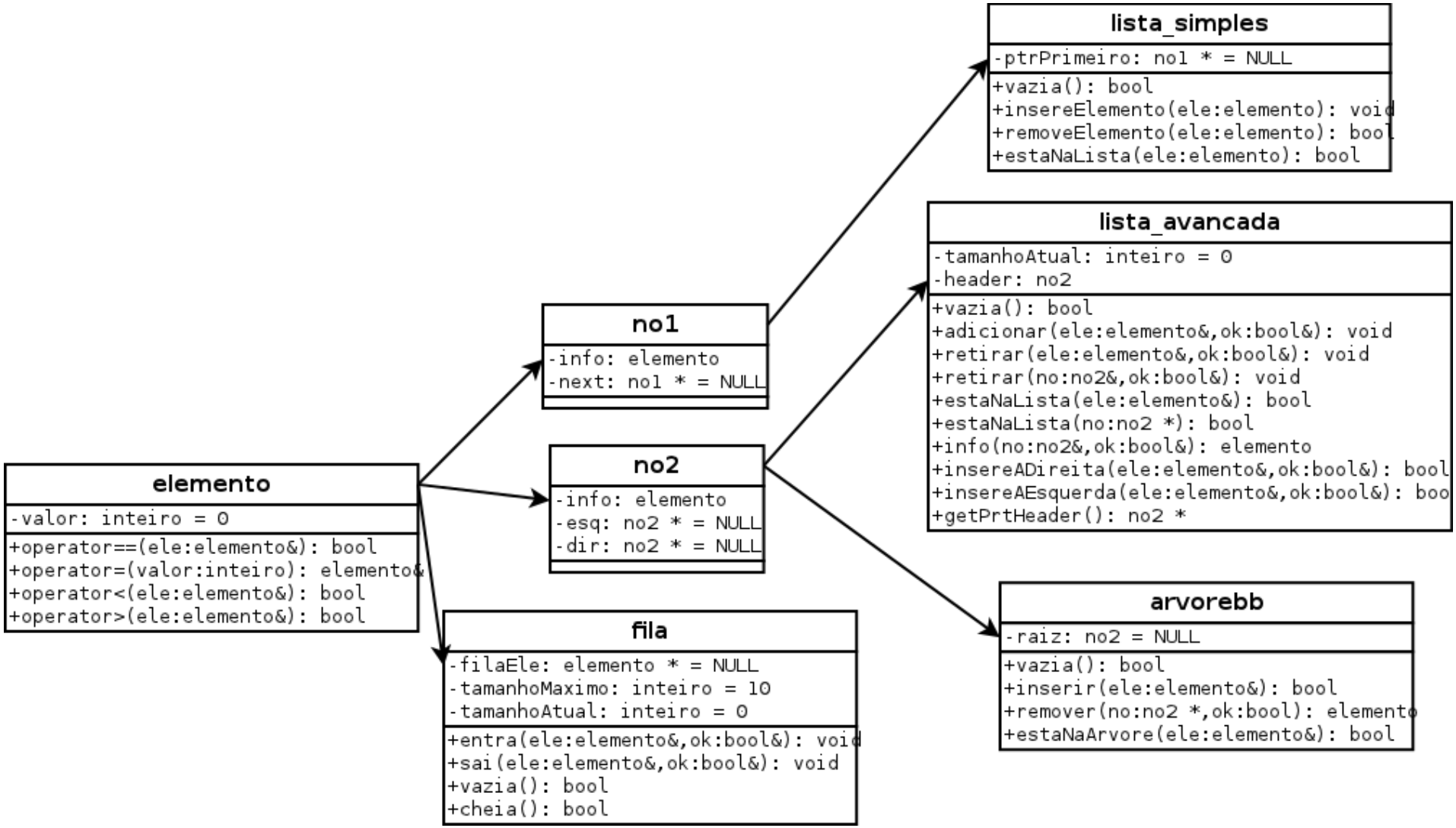
9.1 - Classe ARVORE: insere e remove

9.2 - Classe ARVORE: estaNaArvore

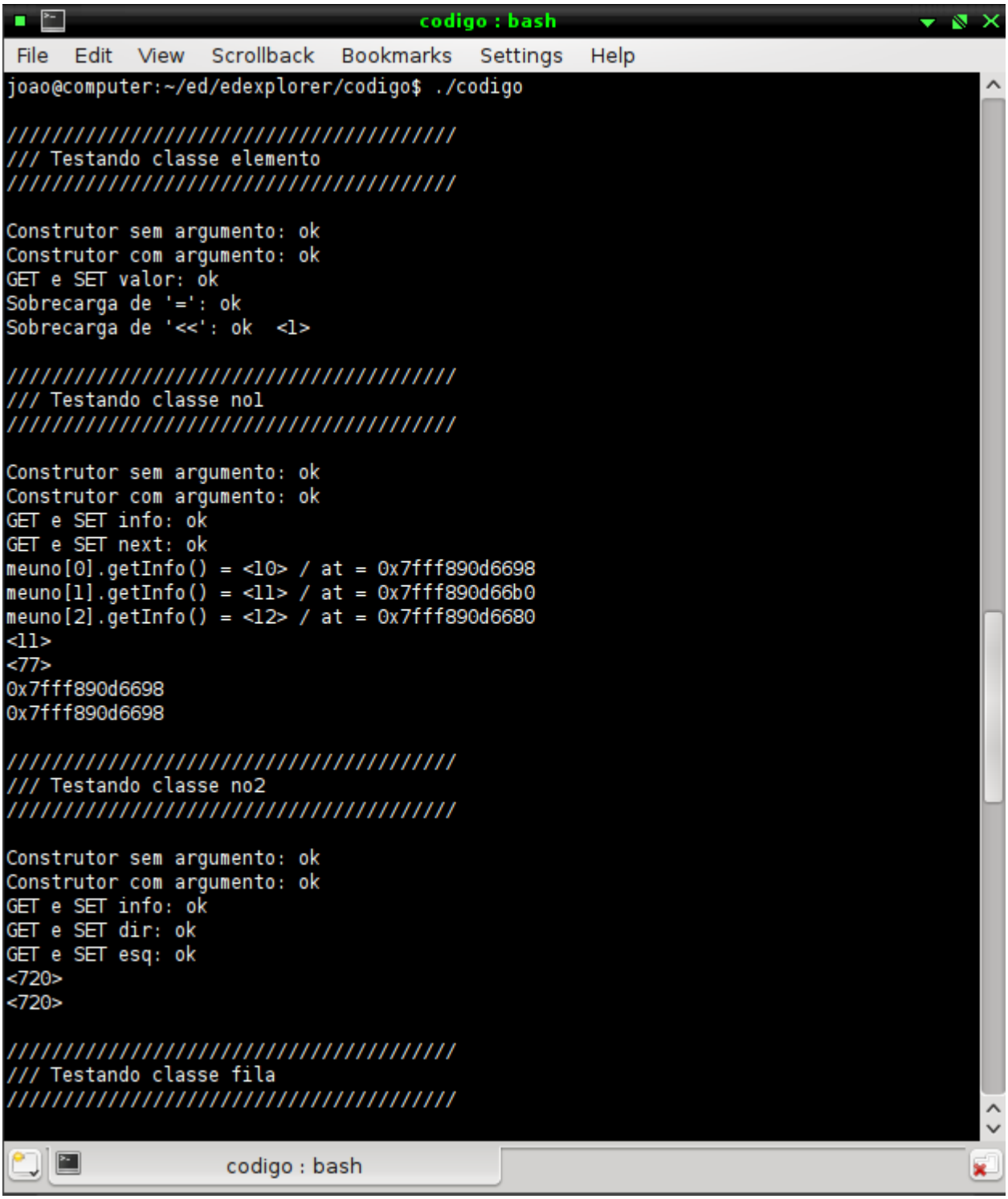
Para cada classe implementada o grupo preparou um conjunto de testes de funcionalidade na função main cujos resultados são impressos na tela.

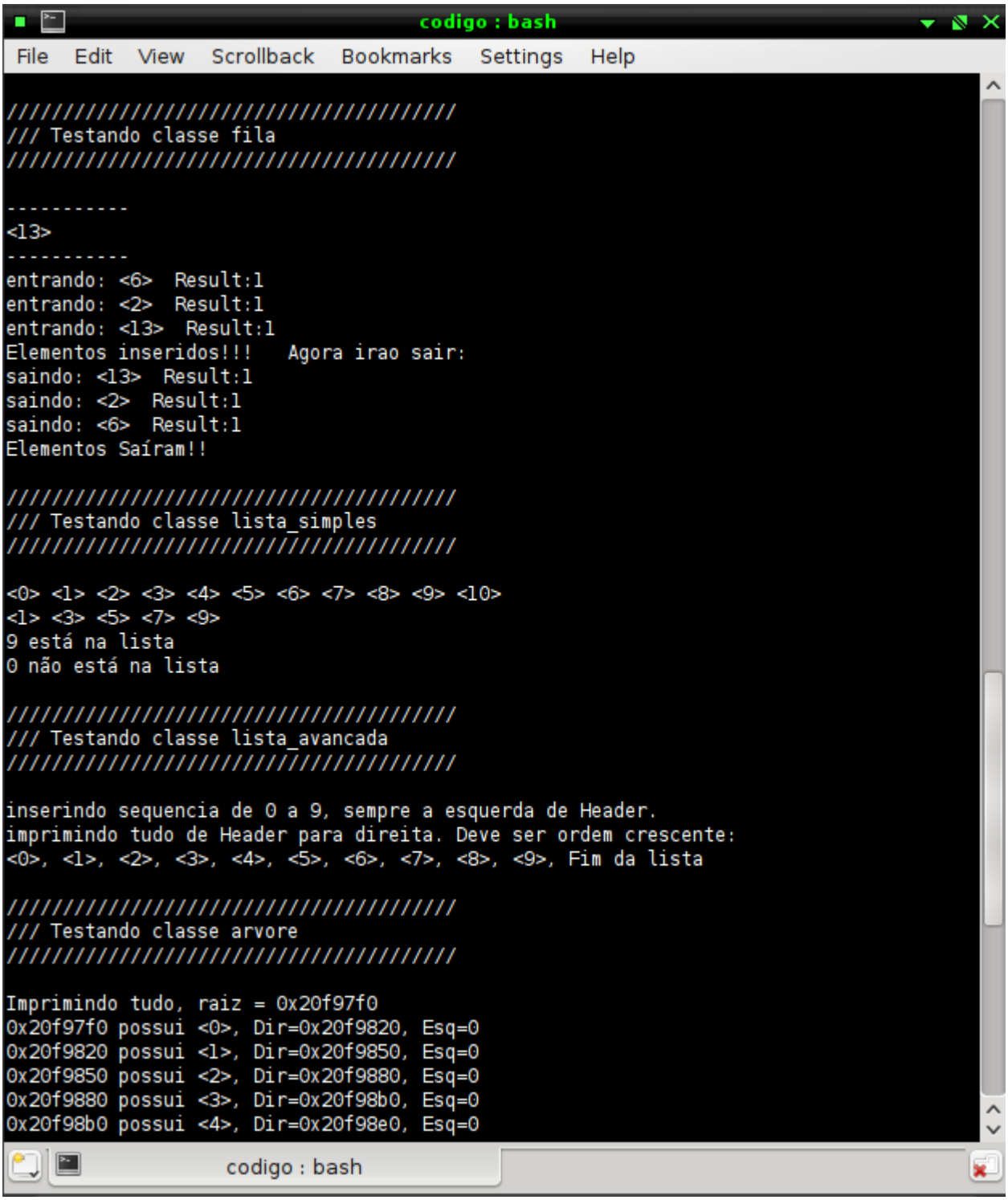
Desenho do Projeto

(na próxima página)

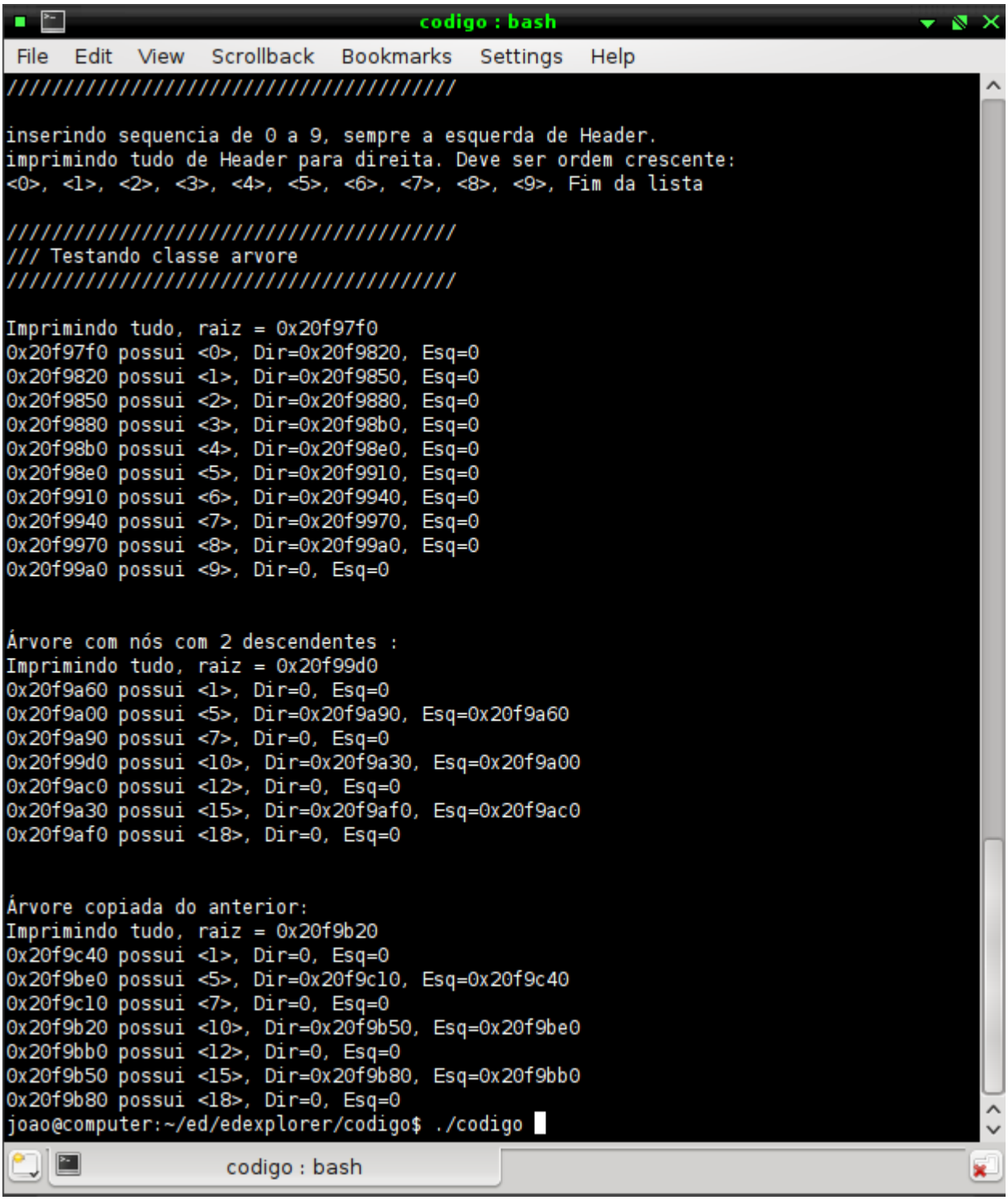

lista_simples
-ptrPrimeiro: no1 * = NULL
+vazia(): bool
+insereElemento(ele:elemento): void
+removeElemento(ele:elemento): bool
+estaNaLista(ele:elemento): bool
lista_avancada
-tamanhoAtual: inteiro = 0
-header: no2
+vazia(): bool
+adicionar(ele:elemento&,ok:bool&): void
+retirar(ele:elemento&,ok:bool&): void
+retirar(no:no2&,ok:bool&): void
+estaNaLista(ele:elemento&): bool
+estaNaLista(no:no2 *): bool
+info(no:no2&,ok:bool&): elemento
+insereADireita(ele:elemento&,ok:bool&): bool
+insereAEsquerda(ele:elemento&,ok:bool&): boo
+getPrtHeader(): no2 *
no1
-info: elemento
-next: no1 * = NULL
no2
-info: elemento
-esq: no2 * = NULL
-dir: no2 * = NULL
elemento
-valor: inteiro = 0
+operator==(ele:elemento&): bool
+operator=(valor:inteiro): elemento&
+operator<(ele:elemento&): bool
+operator>(ele:elemento&): bool
fila
-filaEle: elemento * = NULL
-tamanhoMaximo: inteiro = 10
-tamanhoAtual: inteiro = 0
+entra(ele:elemento&,ok:bool&): void
+sai(ele:elemento&,ok:bool&): void
+vazia(): bool
+cheia(): bool
arvorebb
-raiz: no2 = NULL
+vazia(): bool
+inserir(ele:elemento&): bool
+remover(no:no2 *,ok:bool): elemento
+estaNaArvore(ele:elemento&): bool

Screenshots




codigo : bash
File Edit View Scrollback Bookmarks Settings Help
//
/// Testando classe fila
//

<13>
----
entrando: <6> Result:1
entrando: <2> Result:1
entrando: <13> Result:1
Elementos inseridos!!!   Agora irao sair:
saindo: <13> Result:1
saindo: <2> Result:1
saindo: <6> Result:1
Elementos Saíram!!
//
/// Testando classe lista_simples
//
<0> <1> <2> <3> <4> <5> <6> <7> <8> <9> <10>
<1> <3> <5> <7> <9>
9 está na lista
0 não está na lista
//
/// Testando classe lista_avancada
//
inserindo sequencia de 0 a 9, sempre a esquerda de Header.
imprimindo tudo de Header para direita. Deve ser ordem crescente:
<0>, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, Fim da lista
//
/// Testando classe arvore
//
Imprimindo tudo, raiz = 0x20f97f0
0x20f97f0 possui <0>, Dir=0x20f9820, Esq=0
0x20f9820 possui <1>, Dir=0x20f9850, Esq=0
0x20f9850 possui <2>, Dir=0x20f9880, Esq=0
0x20f9880 possui <3>, Dir=0x20f98b0, Esq=0
0x20f98b0 possui <4>, Dir=0x20f98e0, Esq=0
codigo : bash


codigo : bash
File Edit View Scrollback Bookmarks Settings Help
//
inserindo sequencia de 0 a 9, sempre a esquerda de Header.
imprimindo tudo de Header para direita. Deve ser ordem crescente:
<0>, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, Fim da lista
//
/// Testando classe arvore
//
Imprimindo tudo, raiz = 0x20f97f0
0x20f97f0 possui <0>, Dir=0x20f9820, Esq=0
0x20f9820 possui <1>, Dir=0x20f9850, Esq=0
0x20f9850 possui <2>, Dir=0x20f9880, Esq=0
0x20f9880 possui <3>, Dir=0x20f98b0, Esq=0
0x20f98b0 possui <4>, Dir=0x20f98e0, Esq=0
0x20f98e0 possui <5>, Dir=0x20f9910, Esq=0
0x20f9910 possui <6>, Dir=0x20f9940, Esq=0
0x20f9940 possui <7>, Dir=0x20f9970, Esq=0
0x20f9970 possui <8>, Dir=0x20f99a0, Esq=0
0x20f99a0 possui <9>, Dir=0, Esq=0
Árvore com nós com 2 descendentes :
Imprimindo tudo, raiz = 0x20f99d0
0x20f9a60 possui <1>, Dir=0, Esq=0
0x20f9a00 possui <5>, Dir=0x20f9a90, Esq=0x20f9a60
0x20f9a90 possui <7>, Dir=0, Esq=0
0x20f99d0 possui <10>, Dir=0x20f9a30, Esq=0x20f9a00
0x20f9ac0 possui <12>, Dir=0, Esq=0
0x20f9a30 possui <15>, Dir=0x20f9af0, Esq=0x20f9ac0
0x20f9af0 possui <18>, Dir=0, Esq=0
Árvore copiada do anterior:
Imprimindo tudo, raiz = 0x20f9b20
0x20f9c40 possui <1>, Dir=0, Esq=0
0x20f9be0 possui <5>, Dir=0x20f9c10, Esq=0x20f9c40
0x20f9c10 possui <7>, Dir=0, Esq=0
0x20f9b20 possui <10>, Dir=0x20f9b50, Esq=0x20f9be0
0x20f9bb0 possui <12>, Dir=0, Esq=0
0x20f9b50 possui <15>, Dir=0x20f9b80, Esq=0x20f9bb0
0x20f9b80 possui <18>, Dir=0, Esq=0
joao@computer:~/ed/edexplorer/codigo$ ./codigo
codigo : bash

Conclusão

À partir desse trabalho conseguimos entender como funcionam as estruturas ensinadas no curso e como empregá-las no futuro. Descobrimos que são pivotais para a construção de programas mais portáteis e de código mais reutilizavel.

Esse trabalho é chave no entendimento do curso, pois mostram que esses estruturas facilitam muito o trabalho com dados, desde filas, listas simples e avançadas até árvores binárias.