Linguagens e Ambientes de Programação

Exame de Época Normal 2009/2010 – Proposta de Resolução

1.

A	B	C	D	E
c	a	b	b	b

2. `f : ('a -> int -> 'a) -> 'a -> int -> 'a`

3.

a)
```
let rec countLetter l z =
    match z with
        | [] -> 0
        | Node(c,p,cs)::ts ->
            (if c = l then 1 else 0)
                + countLetter l cs + countLetter l ts
;;
```

b)
```
let rec union z1 z2 =
    match z1, z2 with
        | [],_ -> z2
        | _,[] -> z1
        | Node(c1,p1,cs1)::ts1, Node(c2,p2,cs2)::ts2 ->
            if c1 < c2 then Node(c1,p1,cs1)::union ts1 z2
            else if c1 > c2 then Node(c2,p2,cs2)::union z1 ts2
            else Node(c1,p1 || p2,union cs1 cs2)::union ts1 ts2
;;
```

c)
```
let rec inter z1 z2 =
    match z1, z2 with
        | [],_ -> []
        | _,[] -> []
        | Node(c1,p1,cs1)::ts1, Node(c2,p2,cs2)::ts2 ->
            if c1 < c2 then inter ts1 z2
            else if c1 > c2 then inter z1 ts2
            else let i = inter cs1 cs2 in
                if i = [] && not(p1 && p2) then inter ts1 ts2
                else Node(c1, p1 && p2, i)::inter ts1 ts2
;;
```

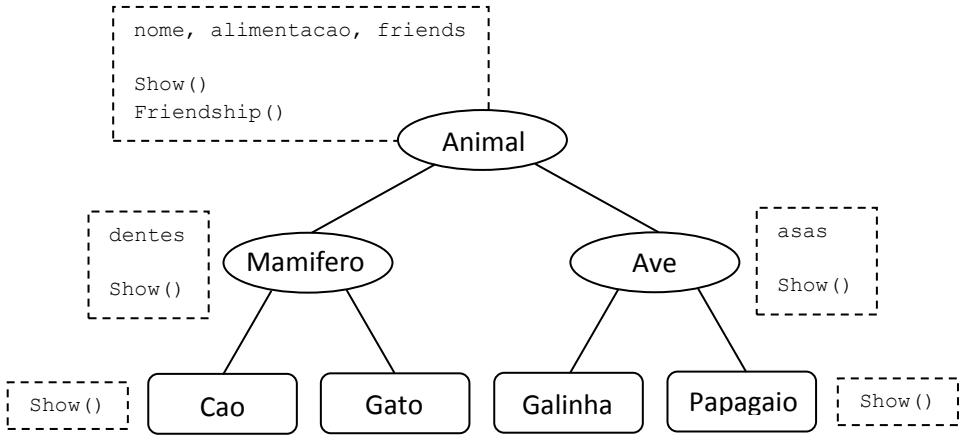
d)
```
let rec sort z =
    match z with
        | [] -> []
        | Node(c,p,cs)::ts ->
            union [Node(c,p,sort cs)] (sort ts)
;;
```

4.

30:	c	0	
	PC	?	
	SL	17	
	DL	22	
	pc	25	
25:	b	0 –> 2	-- F --
	PC	?	
	SL	17	
	DL	17	
	pb	20	
20:	a	0 –> 1	-- main --
	PC	?	
	SL	00	
	DL	13	
	c	0	-- G --
15:	PC	?	
	SL	03	
	DL	08	
	pc	11	
	b	0 –> 2	-- F --
10:	PC	?	
	SL	03	
	DL	03	
	pb	06	
	a	0 –> 1	-- main --
05:	PC	?	
	SL	00	
	DL	00	
	PC	?	-- start --
	SL	?	
00:	DL	?	

5.
```
List Merge(List l1, List l2)
{
    List init = NewNode(0, NULL);     // Nó artificial
    List p = init;

    while((l1 != NULL) && (l2 != NULL))
    {
        if(l1->data < l2->data)
        {
            p = p->next = NewNode(l1->data, NULL);
            l1 = l1->next;
        }

        else if(l1->data > l2->data)
        {
            p = p->next = NewNode(l2->data, NULL);
            l2 = l2->next;
        }

        else
        {
            p = p->next = NewNode(l1->data, NULL);
            l2 = l2->next;
            l1 = l1->next;
        }
    }

    for( ; l1 != NULL; l1 = l1->next)
        p = p->next = NewNode(l1->data, NULL);

    for( ; l2 != NULL; l2 = l2->next)
        p = p->next = NewNode(l2->data, NULL);

    p = init->next;
    free(init);

    return p;
}
```

6.



***.h**

```
#ifndef _Animal_
#define _Animal_

#include <string>
#include <vector>

class Animal;
std::vector<Animal *> all;      // todos os animais

class Animal {
    private:
        std::string nome, alimentacao;
        std::vector<Animal *> friends;
    public:
        Animal(std::string nome, std::string alimentacao,
                                    std::vector<Animal *> friends);
        virtual ~Animal();
        virtual void Show();
        virtual void Friendship();
};

class Mamifero : public Animal {
    private:
        int dentes;
    public:
        Mamifero(std::string nome, std::string alimentacao,
                        int dentes, std::vector<Animal *> friends);
        virtual ~Mamifero;
        void Show();
};

class Cao : public Mamifero {
    private:
        const std::string especie;
    public:
        Cao(std::string nome, std::string alimentacao,
                        int dentes, std::vector<Animal *> friends);
        virtual ~Cao();
        void Show();
};
```

```
class Gato : public Mamifero {
    private:
        const std::string especie;
    public:
        Gato(std::string nome, std::string alimentacao,
                        int dentes, std::vector<Animal *> friends);
        virtual ~Gato();
        void Show();
};

#endif
```

***.cpp**

```
#include "Animal.h"
#include <string>
#include <vector>
#include <iostream>

Animal::Animal(std::string nome, std::string alimentacao,
    std::vector<Animal *> friends) :
        nome(nome), alimentacao(alimentacao) friends(friends) {}
Animal::~Animal() {}

void Animal::Show() {
    std::cout << "Nome: " << nome << "- Alimentação: "
                                            << alimentacao << "- ";
}

void Animal::Friendship() {
  // TODO: implement Friendship()
}

Mamifero::Mamifero(std::string nome, std::string alimentacao,
    int dentes, std::vector<Animal *> friends) :
        Animal(nome, alimentacao, friends), dentes(dentes) {}
Mamifero::~Mamifero() {}

void Mamifero::Show() {
    Animal::Show();
    std::cout << "Dentes: " << dentes << std::endl;
}

Cao::Cao(std::string nome, std::string alimentacao, int dentes,
    std::vector<Animal *> friends) :
        Animal(nome, alimentacao, friends), Mamifero(dentes) {}
Cao::~Cao() {}

void Cao::Show() {
    std::cout << "Cão - ";
    Animal::Show();
    Mamifero::Show();
}

Gato::Gato(std::string nome, std::string alimentacao, int dentes,
    std::vector<Animal *> friends) :
        Animal(nome, alimentacao, friends), Mamifero(dentes) {}
Gato::~Gato() {}

void Gato::Show() { ... }
```