DISCIPLINA IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

GRUPO: CALANGOS DO ACRE

MEMBROS: FRANCISCO DE ASSIS CAMPOS JUNIOR

KAIO HENRIQUE DE SOUSA

ALEX BARBOSA FÉLIX DA SILVA

RELATÓRIO DO PROJETO: PETFERA

Natal, 26/06/2019

1 – DESCRIÇÃO DO TRABALHO:

O projeto PetFera envolve a produção de um protótipo para uma Pet Store, que deve lidar com o banco de animais disponíveis e funcionários desta loja fictícia. Objetivando exercitar e consolidar o aprendizado do paradigma da Programação Orienteda à Objetos, se fez necessário implementar Classes, e através de seus atributos e métodos, executar o programa.

Além das classes pedidas inicialmente (Anfibios, Repteis, Aves e Mamiferos), decidimos incluir também as classes (e suas respectivas classes-filhas): Peixes, Insetos e Aracnideos.

Aracnideo.h

```
#ifndef _ARACNIDEO_H_
#define _ARACNIDEO_H_

#include "Animal.h"
#include "date.h"

using namespace std;

class Aracnideo : public Animal {
  protected:
    int m_total_de_mudas;
    string m_tipo_veneno;
    date m_ultima_muda;
  public:
    Aracnideo();
    Aracnideo(int id, string classe, string nome_cientifico, char sexo,
              double tamanho, string dieta, shared_ptr<Veterinario> veterinario,
              shared_ptr<Tratador> tratador, string nome_batismo, int total_de_mudas,
              string tipo_veneno, int day, int month, int year);
    ~Aracnideo();

    void set_tipo_veneno(string tipo_veneno);
    void set_total_de_mudas(int total_de_mudas);
    void set_ultima_muda(date ultima_muda);
    string get_tipo_veneno();
    int get_total_de_mudas();
    date get_ultima_muda();
    string write();
    string Tipo();

    void inicializar_aracnideo(int id);
    void alterar_aracnideo(string atributo);
  private:
    ostream& print(ostream& os)const;
};

#endif
```

Classe Aracnideo

Inseto.h

```
#ifndef _INSETO_H_
#define _INSETO_H_

#include "Animal.h"
#include "date.h"

using namespace std;

class Inseto : public Animal {
  protected:
    int m_total_de_mudas;
    string m_tipo_metamorfose;
    date m_ultima_muda;
  public:
    Inseto();
    Inseto(int id, string classe, string nome_cientifico, char sexo,
           double tamanho, string dieta, shared_ptr<Veterinario> veterinario,
           shared_ptr<Tratador> tratador, string nome_batismo, int total_de_mudas,
           string tipo_metamorfose, int day, int month, int year);
    ~Inseto();

    void set_tipo_metamorfose(string tipo_metamorfose);
    void set_total_de_mudas(int total_de_mudas);
    void set_ultima_muda(date ultima_muda);
    string get_tipo_metamorfose();
    int get_total_de_mudas();
    date get_ultima_muda();
    string write();
    string Tipo();

    void inicializar_inseto(int id);
    void alterar_inseto(string atributo);
  private:
    ostream& print(ostream& os)const;
};

#endif
```

Classe Inseto

Peixe.h

```h
#ifndef _PEIXE_H
#define _PEIXE_H

#include "Animal.h"

using namespace std;

class Peixe : public Animal{
  protected:
    string m_tipo_agua;

  public:
    Peixe();
    Peixe(int id, string classe, string nome_cientifico, char sexo,
                  double tamanho, string dieta, shared_ptr<Veterinario> veterinario,
                  shared_ptr<Tratador> tratador, string nome_batismo, string tipo_agua);
    ~Peixe();

    void set_tipo_agua(string tipo_agua);
    string get_tipo_agua();
    string write();
    string Tipo();

    void inicializar_peixe(int id);
    void alterar_peixe(string atributo);
  private:
    ostream& print(ostream& os)const;
};

#endif
```

Classe Peixe

Para comportar todas as funcionalidades do programa, foi decidido utilizar uma classe de Controle. Cada um de seus métodos representa uma função do programa.

Controle.h

```
#ifndef CONTROLE_H
#define CONTROLE_H

#include <iostream>
#include <fstream>
#include <map>

#include "Tratamento.h"

using namespace std;

    class Controle{
        private:
            map<int, shared_ptr<Animal>> animais_m;
            map<int, shared_ptr<Funcionario>> funcionarios_m;
            Tratamento tratamento;
            int id;

            void definir_responsavel(shared_ptr<Animal> &bicho, string funcao);
            void criar_animal(shared_ptr<Animal> &bicho, string classe);
            void criar_animal_nativo(shared_ptr<AnimalNativo> &bicho);
            void criar_animal_exotico(shared_ptr<AnimalExotico> &bicho);

        public:
            Controle();
            Controle(map<int, Animal> a, map<int, Funcionario> f);
            ~Controle();

            int is_number(char * number);

            void adicionar_animal();
            void remover_animal();
            void alterar_animal();
            void consultar_animais();
            void consultar_animais_por_funcionario();
            void salvar_alteracoes();

            void adicionar_funcionario();
            void remover_funcionario();
            void alterar_funcionario();
            void consultar_funcionario();
            bool funcionario_valido();
            int definir_id();

};
#endif
```

Classe Controle

O tratamento de exceção ocorre também na classe controle.

```
    //Leitura dos Funcionários
    archive.open("data/Funcionarios.csv");
    if(!(archive.is_open())){
        cerr<<"O arquivo de funcionários não foi aberto!!!"<<endl;
        return;
    }

    try{
        shared_ptr<Funcionario> funcionario;

        while(getline(archive, line)){
            funcionario = tratamento.Tratamento_Construtor_Funcionario(line);
            if(funcionario != NULL)
                funcionarios_m[funcionario->get_id()] = funcionario;
        }
    }
    catch (bad_alloc& ba){
        cerr << "Falha ao alocar memória: " << ba.what() << endl;
    }

    archive.close();

    //Leitura dos Animais
    archive.open("data/Animais.csv");
    if(!(archive.is_open())){
        cerr<<"O arquivo de animais não foi aberto!!!"<<endl;
        return;
    }

    try{
        shared_ptr<Animal> bicho;

        while(getline(archive, line)){
            bicho = tratamento.Tratamento_Construtor_Animal(line, funcionarios_m);
            if(bicho != NULL)
                animais_m[bicho->get_m_id()] = bicho;
        }
    }
    catch (bad_alloc& ba){
        cerr << "Falha ao alocar memória: " << ba.what() << endl;
    }

    archive.close();
};
```

Tratamento de exceção

E para auxiliar o Controle no caso de tratamentos, foram criadas as classes Tratamento e TratamentoInput.

Tratamento.h

```h
#ifndef _TRATAMENTO_H_
#define _TRATAMENTO_H_

#include <string>
#include <iostream>
#include <memory>
#include <map>
#include "AnfibioDomestico.h"
#include "AnfibioExotico.h"
#include "AnfibioNativo.h"
#include "AracnideoDomestico.h"
#include "AracnideoExotico.h"
#include "AracnideoNativo.h"
#include "AveDomestica.h"
#include "AveExotica.h"
#include "AveNativa.h"
#include "InsetoDomestico.h"
#include "InsetoExotico.h"
#include "InsetoNativo.h"
#include "MamiferoDomestico.h"
#include "MamiferoExotico.h"
#include "MamiferoNativo.h"
#include "ReptilDomestico.h"
#include "ReptilExotico.h"
#include "ReptilNativo.h"
#include "PeixeDomestico.h"
#include "PeixeExotico.h"
#include "PeixeNativo.h"
#include "Tratador.h"
#include "Veterinario.h"

using namespace std;

class Tratamento{
    private:

    public:
        shared_ptr<Animal> Tratamento_Construtor_Animal(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AnfibioDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AnfibioExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AnfibioNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AracnideoDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AracnideoExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AracnideoNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AveDomestica(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AveExotica(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_AveNativa(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_InsetoDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_InsetoExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_InsetoNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_MamiferoDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_MamiferoExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_MamiferoNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_ReptilDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_ReptilExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_ReptilNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_PeixeDomestico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_PeixeExotico(string line, map<int , shared_ptr<Funcionario>> &funcionarios);
        shared_ptr<Animal> Tratamento_PeixeNativo(string line, map<int , shared_ptr<Funcionario>> &funcionarios);

        shared_ptr<Funcionario> Tratamento_Construtor_Funcionario(string line);
        shared_ptr<Funcionario> Tratamento_Tratador(string line);
        shared_ptr<Funcionario> Tratamento_Veterinario(string line);

        string * Tratamento_Data(string line);


};

#endif
```

Classe Tratamento

TratamentoInput.h

```
#ifndef _TRATAMENTOINPUT_H_
#define _TRATAMENTOINPUT_H_

#include <iostream>
#include <string>
#include "ExcecoesInput.h"

using namespace std;

class TratamentoInput{

    public:
        char * inputChar();
        double is_number(char * number);
        string inputString();
        int inputInt();
        double inputDouble();
};

#endif
```

Classe TratamentoInput

2 – EXECUÇÃO DO PROGRAMA

Loja:

Para executar o programa, primeiramente será necessário efetuar seu download no repositório git:

https://github.com/juniorCampos0/PetFera.git

```
File  Edit  View  Search  Terminal  Help
magusk@DEADSIDE:~$ git clone https://github.com/juniorCampos0/PetFera
Cloning into 'PetFera'...
remote: Enumerating objects: 766, done.
remote: Counting objects: 100% (766/766), done.
remote: Compressing objects: 100% (439/439), done.
remote: Total 2024 (delta 503), reused 559 (delta 307), pack-reused 1258
Receiving objects: 100% (2024/2024), 21.31 MiB | 374.00 KiB/s, done.
Resolving deltas: 100% (1480/1480), done.
magusk@DEADSIDE:~$
```

Após isso, acessamos a pasta do programa:

```
magusk@DEADSIDE:~$ cd PetFera/
magusk@DEADSIDE:~/PetFera$
```

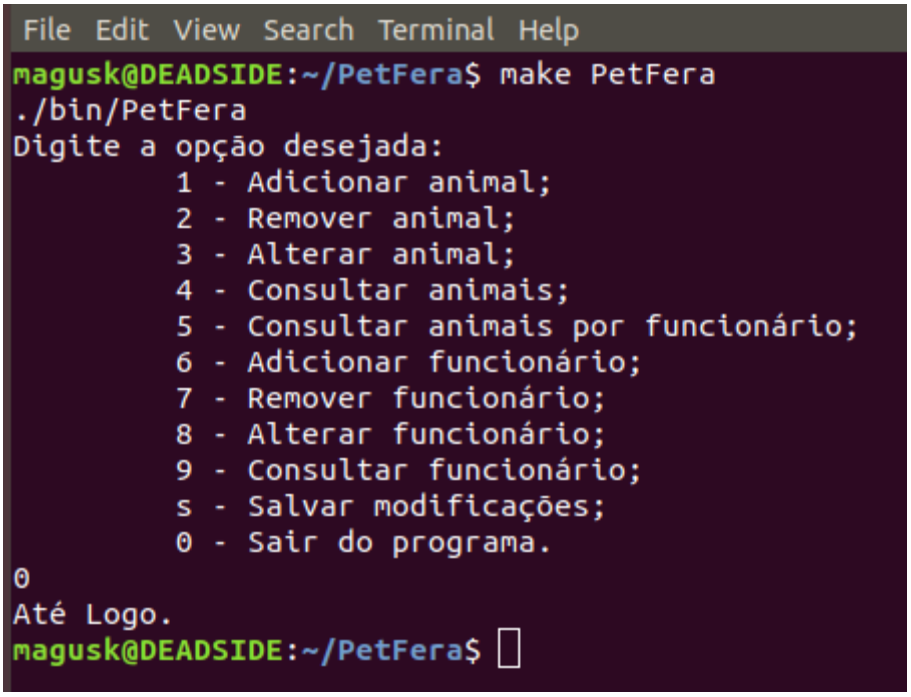
Executamos o makefile:

```
magusk@DEADSIDE:~/PetFera$ make
g++ -O0 -Wall -g -std=c++11 -I include  src/Funciona
g++ -O0 -Wall -g -std=c++11 -I include  src/Animal.c
g++ -O0 -Wall -g -std=c++11 -I include  src/Tratador
```

E então executamos o programa com o comando “make PetFera” ou “./bin/PetFera”:

```
File  Edit  View  Search  Terminal  Help
magusk@DEADSIDE:~/PetFera$ make PetFera
./bin/PetFera
Digite a opção desejada:
        1 - Adicionar animal;
        2 - Remover animal;
        3 - Alterar animal;
        4 - Consultar animais;
        5 - Consultar animais por funcionário;
        6 - Adicionar funcionário;
        7 - Remover funcionário;
        8 - Alterar funcionário;
        9 - Consultar funcionário;
        s - Salvar modificações;
        0 - Sair do programa.
```

A partir deste ponto, basta escolher a opção desejada e utilizar o programa normalmente:

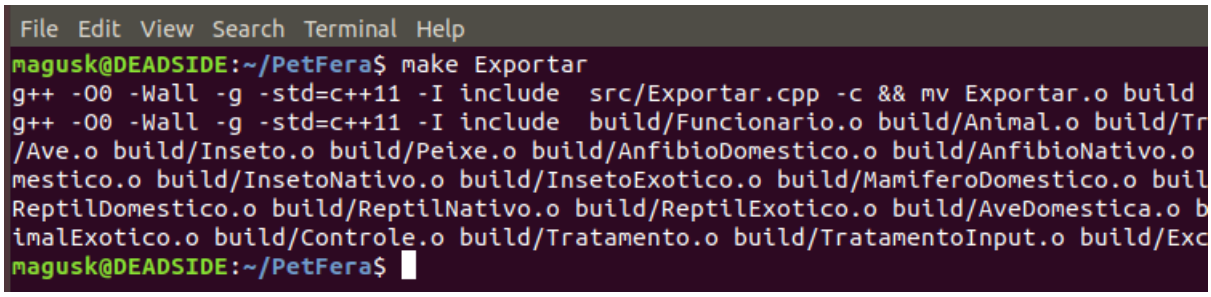


Para maiores detalhes, assista ao vídeo de execução completa do programa em:

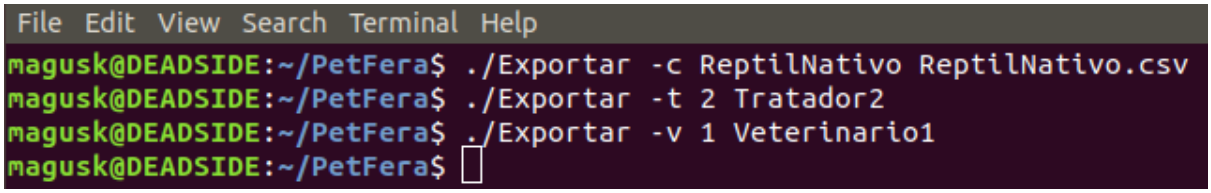
https://www.youtube.com/watch?v=QQMaffrYIYM

Exportador:

Para criar o Exportador, executamos o makefile com o comando “make Exportar”:



E para executar o Exportador, utilizamos o comando “./Exportar”, lembrando que o Exportador funciona através de linhas de comando:



-c NomedaClasse NomedoArquivo.csv: gera um arquivo NomedoArquivo.csv com todos os animais pertencentes à classe NomedaClasse.

-t 2 Tratador2: gera um arquivo Tratador2.csv com todos os animais que tem como tratador aquele de ID 2.

-v 1 Veterinario1: gera um arquivo Veterinario1.csv com todos os animais que tem como veterinário aquele de ID 1.

3 – DIFICULDADES

Definir a melhor forma de gravar as informações em arquivos e efetuar sua leitura: o grupo encontrou dificuldade em definir como gravar e ler as informações nos arquivos e optou por utilizar o método Type() para definir o tipo da informação e assim facilitar a identificação no momento da leitura.

Adicionar animais: no caso da adição de animais, precisávamos fazer o tratamento de entrada do usuário, para impedir a entrada de informações incompatíveis com o atributo. Foi decidido então criar a classe de TratamentoInput referenciada acima, no item 1.

Remover funcionário: Poderia acontecer a remoção de um funcionário com um animal associado, então decidimos impedir o usuário de fazê-lo neste caso especifico e é alertado a necessidade de mudar os funcionários associados desses animais.