|  |  |  |  |
| --- | --- | --- | --- |
| **Group Members:** | S**tudent ID’s:** | **Contributions to the project:** | **Signatures to confirm contribution:** |
| Matthew Newberry | 6420459 | Implementation(33%), Preliminary Design(33%), Results(33%) | C:\Users\rosab\AppData\Local\Microsoft\Windows\INetCache\Content.Word\25509024_1950789068282861_664863221_o.jpg |
| Rosabelle Armstead | 6402882 | Implementation(33%), Preliminary Design(33%), Results(33%) | https://dl2.pushbulletusercontent.com/bNuAtcSZ0Gy6i9eLZ3r895eGNPKHEgjv/DSC_1323.JPG |
| Harry Pegrum | 6419920 | Implementation(33%), Preliminary Design(33%), Results(33%) | C:\Users\rosab\AppData\Local\Microsoft\Windows\INetCache\Content.Word\25508747_1309412235829602_1343383534_o.png |

**Explanation of Implemented Code:**

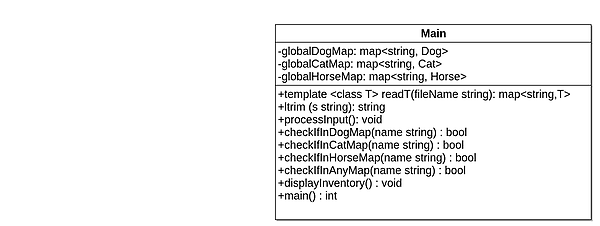
The program we have created first outputs the entire inventory of animals, as read from the csv files included in the program, it then takes an input from the user which it splits into an animal type and a name. It then displays the paternal tree of the animal if it can be found in the requested animal type, otherwise it outputs a message to let the user know the requested animal could not be found.

The csv files are first read into a map for each type of animal, so there is a dog map, a cat map and a horse map. These maps store the name of the animal as the key and the animal object as the value. When a user inputs a value the correct maps are searched for the animal, if they are found then we use a father pointer in the animal class to point to the next animal in the paternal tree. Following this list of pointers eventually ends with an animal with no father and thus the paternal tree terminates. We have also used template methods when processing any of the maps as each map has a different value date type depending on the animal. This has reduced the amount of written code in the Main class.

**Revised Class Hierarchy and Main class:**

The class hierarchy was not changed from the design.

We added several new methods to the Main class to separate each step of the process and to make it clearer as to what each method does. This also made it easier to work on the implementation as a group.

****

The Main class now has the following attributes:

* globalDogMap (type: map<string, Dog>) – A map of Dog objects where the key is the name of the dog
* globalCatMap (type: map<string, Cat>) – A map of Cat objects where the key is the name of the cat
* globalHorseMap (type: map<string, Horse>) – A map of Horse objects where the key is the name of the horse

The Main class also now has the following methods:

* template <class T> readT(input type: string) (return type: map<string, T>) – Takes the file name as an input and parses the csv into a map of objects of the relevant class. Returns the map of objects.
* template <class T> processAnimal (input type: string, map<string, T>) (return type: void) – takes the name of the animal and the relevant map for the relevant class and outputs the paternal family tree of the animal.
* ltrim(input type: string) (return type: string) – Trims the first two characters of the input string so that the name of the animal can be used.
* processInput() (return type: void) – Checks which animal inventory the user is searching in.
* checkIfInDogMap(input type: string) (return type: bool) – Checks if the animal searched for in the dog inventory exists in the dog inventory. An error message is displayed if it does not.
* checkIfInCatMap(input type: string) (return type: bool) - Checks if the animal searched for in the cat inventory exists in the cat inventory. An error message is displayed if it does not.
* checkIfInHorseMap(input type: string) (return type: bool) - Checks if the animal searched for in the horse inventory exists in the horse inventory. An error message is displayed if it does not.
* checkIfInAnyMap(input type: string) (return type: void) - Checks if the animal searched for exists in any inventory. An error message is displayed if it does not.

**Code:**

*Aesthetics.cpp:*

The Aesthetics.cpp file defines the behaviour of aesthetics, including the constructor and the getters for the colour, ear type, height and tail colour.

*/\**

*\* Aesthetics.cpp*

*\**

*\* Defines the behaviour of Aesthetics such as colour and ear type.*

*\**

*\*/*

#include <string>

#include "Aesthetics.h"

using namespace std;

*/\**

*\* Aesthetics constructor*

*\*/*

Aesthetics::Aesthetics(string a\_colour, string a\_earType, string a\_height, string a\_tailColour)

{

colour = a\_colour;

earType = a\_earType;

height = a\_height;

tailColour = a\_tailColour;

}

*/\**

*\* Getter for colour*

*\*/*

string Aesthetics::getColour() {

return colour;

}

*/\**

*\* Getter for ear type*

*\*/*

string Aesthetics::getEarType() {

return earType;

}

*/\**

*\* Getter for height*

*\*/*

string Aesthetics::getHeight() {

return height;

}

*/\**

*\* Getter for tail colour*

*\*/*

string Aesthetics::getTailColour() {

return tailColour;

}

*Animal.cpp:*

The Animal.cpp file defines the behaviour of an animal, including the constructor and the getters for the breed, name, father pointer and mum’s name.

*/\**

*\* Animal.cpp*

*\**

*\* Defines the behaviour of an Animal object.*

*\**

*\*/*

#include "Animal.h"

#include <string>

using namespace std;

*/\**

*\* Animal constructor*

*\*/*

Animal::Animal(string a\_breed, string a\_name, string a\_colour, string a\_earType, string a\_height, string a\_tailColour, Animal \*a\_animal,string a\_mumsName) : Aesthetics(a\_colour, a\_earType, a\_height, a\_tailColour)

{

breed = a\_breed;

name = a\_name;

animal = a\_animal;

mumsName = a\_mumsName;

}

*/\**

*\* Getter for breed*

*\*/*

string Animal::getBreed() {

return breed;

}

*/\**

*\* Getter for name*

*\*/*

string Animal::getName() {

return name;

}

*/\**

*\* Getter for father*

*\*/*

Animal\* Animal::getDad() {

return animal;

}

*/\**

*\* Getter for mum’s name*

*\*/*

string Animal::getMumsName() {

return mumsName;

}

*Cat.cpp:*

The Cat.cpp file defines the behaviour of a cat, including the constructor.

*/\**

*\* Cat.cpp*

*\**

*\* Defines the behaviour of a Cat object.*

*\**

*\*/*

#include "Cat.h"

#include <string>

using namespace std;

*/\**

*\* Cat constructor*

*\*/*

Cat::Cat(string a\_breed, string a\_name, string a\_colour, string a\_earType, string a\_height, string a\_tailColour, Animal \*a\_dad, string a\_mumsName) : Animal(a\_breed, a\_name, a\_colour, a\_earType, a\_height, a\_tailColour, a\_dad, a\_mumsName)

{

}

*Dog.cpp:*

The Dog.cpp file defines the behaviour of a dog, including the constructor.

*/\**

*\* Dog.cpp*

*\**

*\* Defines the behaviour of a Dog object.  
\**

*\*/*

#include "Dog.h"

#include <string>

using namespace std;

*/\**

*\* Dog constructor*

*\*/*

Dog::Dog(string a\_breed, string a\_name, string a\_colour, string a\_earType, string a\_height, string a\_tailColour, Animal \*a\_dad, string a\_mumsName) : Animal(a\_breed, a\_name, a\_colour, a\_earType, a\_height, a\_tailColour, a\_dad, a\_mumsName)

{

}

*Horse.cpp:*

The Horse.cpp file defines the behaviour of a horse, including the constructor.

*/\**

*\* Horse.cpp*

*\**

*\* Defines the behaviour of a Horse object.  
\*  
\*/*

#include "Horse.h"

#include <string>

using namespace std;

*/\**

*\* Horse constructor*

*\*/*

Horse::Horse(string a\_breed, string a\_name, string a\_colour, string a\_earType, string a\_height, string a\_tailColour, Animal \*a\_dad, string a\_mumsName) : Animal(a\_breed, a\_name, a\_colour, a\_earType, a\_height, a\_tailColour, a\_dad, a\_mumsName)

{

}

*main.cpp:*

The main.cpp file displays the inventory of animals, takes the user input and process it, and outputs the paternal family tree of the animal chosen by the user.

*/\* main.cpp \*/*

#include "Dog.h"

#include "Cat.h"

#include "Horse.h"

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <map>

#include <iomanip>

using namespace std;

*/\* function declarations \*/*

template <class T> map<string,T> readT(string fileName);

template <class T> void processAnimal(string inputAnimal, map<string, T> map);

string ltrim(string s, const char\* t);

void processInput();

bool checkIfInDogMap(string name);

bool checkIfInCatMap(string name);

bool checkIfInHorseMap(string name);

void checkIfInAnyMap(string name);

*/\* global map declarations \*/*

map<string, Dog> globalDogMap;

map<string, Cat> globalCatMap;

map<string, Horse> globalHorseMap;

*/\**

*\* Reads a csv file into a map of T class objects by splitting on commas within*

*\* a line for the class parameters, and using each line as a separate object.*

*\*/*

template <class T> map<string,T> readT(string fileName)

{

string param[8]; *// array for parameters of T object*

map<string, T> tMap;

ifstream infile("files/" + fileName); *// get file from correct directory*

if (infile.is\_open())

{

string line;

while(getline(infile, line))

{

stringstream ss(line);

string item;

int i = 0;

while(getline(ss,item, ','))

{

param[i] = item; *// populate parameter array from line on comma split*

i++;

}

if (tMap.find(param[6]) == tMap.end()) *// if object has no father, set null pointer*

{

T newT(param[0], param[1], param[2], param[3], param[4], param[5],

NULL, param[7]); // null pointer if no father

tMap.insert(pair<string, T> (param[1], newT));

}

else *// if object has father, set pointer to father*

{

Animal \*fatherPointer = &tMap.at(param[6]); *// pointer to father object*

T newT(param[0], param[1], param[2], param[3], param[4], param[5],

fatherPointer, param[7]);

tMap.insert(pair<string, T> (param[1], newT));

}

}

}

else

{

cout << "There was an error reading the file!" << endl; *// error checking for if the file cannot be read*

}

return tMap;

}

*/\**

*\* returns animal name string by trimming first 2 characters of string s*

*\*/*

string ltrim(string s)

{

return s.erase(0, 2);

}

*/\**

*\* Compares first letter of user input to find which map to search for an animal.*

*\* Runs ltrim function on user input to get animal name from user input.*

*\* Runs error checking functions in case animal does no exist in given map, or there*

*\* is no map for the user input.*

*\*/*

void processInput()

{

string inputAnimal = "";

cout << "Enter the first letter of the animal group and the name of the specified one to find its paternal tree: ";

getline(cin, inputAnimal);

*// Checks if input was a character followed by a space followed by a name*

if(inputAnimal[1]!=' ') {

processInput();

} else {

if (inputAnimal[0] == 'd') *// if user inputs d, dog map is checked*

{

inputAnimal = ltrim(inputAnimal);

if(!checkIfInDogMap(inputAnimal)) *// if dog doesn't exist in map, error message*

{

cout << inputAnimal << " was not found within the \e[mDogs!" << endl;

}

}

else if (inputAnimal[0] == 'c') *// if user inputs c, cat map is checked*

{

inputAnimal = ltrim(inputAnimal);

if(!checkIfInCatMap(inputAnimal)) *// if cat doesn't exist in map, error message*

{

cout << inputAnimal << " was not found within the \e[mCats!" << endl;

}

}

else if (inputAnimal[0] == 'h') *// if user inputs h, horse map is checked*

{

inputAnimal = ltrim(inputAnimal);

if(!checkIfInHorseMap(inputAnimal)) *// if horse doesn't exist in map, error message*

{

cout << inputAnimal << " was not found within the \e[mHorses!" << endl;

}

}

else if (inputAnimal[0] == 'a') *// if user inputs a, all maps are checked*

{

inputAnimal = ltrim(inputAnimal);

try *// try catch so if animal doesn't exist in any map, error message*

{

checkIfInAnyMap(inputAnimal);

}

catch (const char\* msg)

{

cerr << inputAnimal << msg << endl;

}

}

else *// if user inputs anything else, there is no map so error message*

{

cout << "Not an animal! Enter a to search all, c to search cats, d to search dogs or h to search horses." << endl;

}

}

}

*/\**

*\* Returns true if user input name is found in dog map*

*\* If true, runs processAnimal function for dog in the dog map*

*\*/*

bool checkIfInDogMap(string name)

{

try

{

globalDogMap.at(name);

processAnimal<Dog>(name, globalDogMap);

return true;

}

catch (const out\_of\_range& e)

{

return false;

}

}

*/\**

*\* Returns true if user input name is found in cat map*

*\* If true, runs processAnimal function for cat in the cat map*

*\*/*

bool checkIfInCatMap(string name)

{

try

{

globalCatMap.at(name);

processAnimal<Cat>(name, globalCatMap);

return true;

}

catch (const out\_of\_range& e)

{

return false;

}

}

*/\**

*\* Returns true if user input name is found in horse map*

*\* If true, runs processAnimal function for horse in the horse map*

*\*/*

bool checkIfInHorseMap(string name)

{

try

{

globalHorseMap.at(name);

processAnimal<Horse>(name, globalHorseMap);

return true;

}

catch (const out\_of\_range& e)

{

return false;

}

}

*/\**

*\* Throws exception if user input name is not found in any map*

*\*/*

void checkIfInAnyMap(string name)

{

if(checkIfInDogMap(name)) {}

else if (checkIfInCatMap(name)) {}

else if (checkIfInHorseMap(name)) {}

else

{

throw " was not found within any inventory!";

}

}

*/\**

*\* Takes user input animal and map for given animal and displays the paternal tree*

*\* of the given animal. Uses father pointer to get address location for father of*

*\* animal.*

*\*/*

template <class T> void processAnimal(string inputAnimal, map<string,T> tMap)

{

try *// try catch for if the animal has no father, display default tree*

{

Animal animal = tMap.at(inputAnimal);

cout << "Paternal tree of " << inputAnimal << ":" << endl;

cout << inputAnimal << " <-- ";

*/\* while loop to get all fathers until an animal with no father \*/*

while(tMap.at(animal.getName()).getDad()!=0)

{

cout << animal.getDad()->getName() << " <-- ";

animal = tMap.at(animal.getDad()->getName());

}

}

catch (const out\_of\_range& e)

{

cout << inputAnimal << " <-- ";

}

cout << "[END]" << endl;

}

*/\**

*\* Displays formatted output of all animals by iterating through the maps and outputting*

*\* the relevant attributes.*

*\*/*

void displayInventory()

{

cout << "There are " << globalDogMap.size() << " dog(s), " << globalCatMap.size() << " cat(s) and " << globalHorseMap.size() << " horse(s) in the inventory, which are:" << endl;

*/\* column titles formatting \*/*

int widthCounter = 91;

cout << left;

cout << setw(10) << "Name" << setw(10) << "Group" << setw(10) << "Breed" << setw(10)

<< "Colour" << setw(12) << "Ear Type" << setw(10) << "Height" << setw(16) << "Tail Colour" << setw(10) << "Dad" << setw(3) << "Mum" << endl;

for (int i = 0; i <= widthCounter; i++)

{

cout << "-";

}

cout << endl;

*/\* display for dogs \*/*

map<std::string, Dog>::iterator dogIt;

for (dogIt = globalDogMap.begin(); dogIt != globalDogMap.end(); dogIt++)

{

Dog currentDog = dogIt->second;

string currentDogMum = "N/A";

string currentDogDad = "N/A";

if (currentDog.getDad() != 0)

{

currentDogMum = currentDog.getMumsName();

currentDogDad = currentDog.getDad()->getName();

}

cout << left;

cout << setw(10) << currentDog.getName() << setw(10) << "Dog" << setw(10) << currentDog.getBreed() << setw(10) << currentDog.getColour() << setw(12) << currentDog.getEarType() <<

setw(10) << currentDog.getHeight() << setw(16) << currentDog.getTailColour() <<

setw(10) << currentDogDad << currentDogMum << endl;

}

cout << endl;

*/\* display for cats \*/*

map<std::string, Cat>::iterator catIt;

for (catIt = globalCatMap.begin(); catIt != globalCatMap.end(); catIt++)

{

Cat currentCat = catIt->second;

string currentCatMum = "N/A";

string currentCatDad = "N/A";

if (currentCat.getDad() != 0)

{

currentCatMum = currentCat.getMumsName();

currentCatDad = currentCat.getDad()->getName();

}

cout << left;

cout << setw(10) << currentCat.getName() << setw(10) << "Cat" << setw(10) <<

currentCat.getBreed() << setw(10) << currentCat.getColour() << setw(12) << currentCat.getEarType() <<

setw(10) << currentCat.getHeight() << setw(16) << currentCat.getTailColour() <<

setw(10) << currentCatDad << currentCatMum << endl;

}

cout << endl;

*/\* display for horses \*/*

map<std::string, Horse>::iterator it;

for (it = globalHorseMap.begin(); it != globalHorseMap.end(); it++)

{

Horse currentHorse = it->second;

string currentHorseMum = "N/A";

string currentHorseDad = "N/A";

if (currentHorse.getDad() != 0)

{

currentHorseMum = currentHorse.getMumsName();

currentHorseDad = currentHorse.getDad()->getName();

}

cout << left;

cout << setw(10) << currentHorse.getName() << setw(10) << "Horse" << setw(10) <<

currentHorse.getBreed() << setw(10) << currentHorse.getColour() << setw(12) << currentHorse.getEarType() <<

setw(10) << currentHorse.getHeight() << setw(16) << currentHorse.getTailColour() <<

setw(10) << currentHorseDad << currentHorseMum << endl;

}

}

*/\**

*\* Entry point for the program. Runs the readT function to create all animal maps. Runs*

*\* the displayInventory function to display all animals. Continues running the program*

*\* until force quit.*

*\*/*

int main()

{

globalDogMap = readT<Dog>("dogs.csv");

globalCatMap = readT<Cat>("cats.csv");

globalHorseMap = readT<Horse>("horses.csv");

displayInventory();

while (true)

{

processInput();

}

}

*Aesthetics.h:*

The Aesthetics.h file defines the properties of the aesthetics class, including it’s methods and attributes. A header guard is used to prevent conflicting redefinitions of the same class.

*/\**

*\* Aesthetics.h*

*\**

*\* Defines the properties of the Aesthetics class.*

*\*  
\*/*

#ifndef AESTHETICS\_H

#define AESTHETICS\_H

#include <string>

*/\**

*\* Methods and attributes of the Aesthetics class.*

*\*/*

class Aesthetics {

public:

Aesthetics(std::string colour, std::string earType, std::string height, std::string tailColour);

std::string getColour();

std::string getEarType();

std::string getHeight();

std::string getTailColour();

private:

std::string colour;

std::string earType;

std::string height;

std::string tailColour;

};

#endif /\* AESTHETICS\_H \*/

*Animal.h:*

The Animal.h file defines the properties of the animal class, including it’s methods and attributes. A header guard is used to prevent conflicting redefinitions of the same class.

*/\**

*\* Animal.h*

*\**

*\* Defines the properties of the Animal class.*

*\**

*\*/*

#ifndef ANIMAL\_H

#define ANIMAL\_H

#include <string>

#include "Aesthetics.h"

*/\**

*\* Methods and attributes of the Animal class. Inherits from the Aesthetics class.*

*\*/*

class Animal : public Aesthetics {

public:

Animal(std::string breed, std::string name, std::string colour, std::string earType, std::string height, std::string tailColour, Animal \*animal, std::string mumsName);

std::string getBreed();

std::string getName();

Animal\* getDad();

std::string getMumsName();

private:

std::string breed;

std::string name;

Animal \*animal;

std::string mumsName;

};

#endif /\* ANIMAL\_H \*/

*Cat.h:*

The Cat.h file defines the properties of a cat object, including it’s methods and attributes. A header guard is used to prevent conflicting redefinitions of the same class.

*/\**

*\* Cat.h*

*\**

*\* Defines the properties of a Cat object.  
\*  
\*/*

#ifndef CAT\_H

#define CAT\_H

#include "Animal.h"

#include <string>

*/\**

*\* Methods and attributes of a Cat object. Inherits from the Animal class.*

*\*/*

class Cat: public Animal{

public:

Cat(std::string breed, std::string name, std::string colour, std::string earType, std::string height, std::string tailColour, Animal \*dad, std::string mumsName);

};

#endif /\* CAT\_H \*/

*Dog.h:*

The Dog.h file defines the properties of a dog object, including it’s methods and attributes. A header guard is used to prevent conflicting redefinitions of the same class.

*/\**

*\* Dog.h*

*\**

*\* Defines the properties of a Dog object.*

*\*  
\*/*

#ifndef DOG\_H

#define DOG\_H

#include "Animal.h"

#include <string>

*/\**

*\* Methods and attributes of a Dog object. Inherits from the Animal class.*

*\*/*

class Dog: public Animal{

public:

Dog(std::string breed, std::string name, std::string colour, std::string earType, std::string height, std::string tailColour, Animal \*dad, std::string mumsName);

};

#endif /\* DOG\_H \*/

*Horse.h:*

The Horse.h file defines the properties of a horse object, including it’s methods and attributes. A header guard is used to prevent conflicting redefinitions of the same class.

*/\**

*\* Horse.h*

*\**

*\* Defines the properties of a Horse object.*

*\**

*\*/*

#ifndef HORSE\_H

#define HORSE\_H

#include "Animal.h"

#include <string>

*/\**

*\* Methods and attributes of a Horse object. Inherits from the Animal class.*

*\*/*

class Horse : public Animal{

public:

Horse(std::string breed, std::string name, std::string colour, std::string earType, std::string height, std::string tailColour, Animal \*dad, std::string mumsName);

};

#endif /\* HORSE\_H \*/