# Animals Competition Project (C++ version)

In this project, you will expand Hare and Tortoise Competition Project to simulate a competition among any animals and no longer be restricted to hare and tortoise any more.

You need to apply the concepts of inheritance and polymorphism. See slides for introduction of these topics, posted in blackboard, you can read more on P358-359 in the textbook.

**Motivation for inheritance**: when we design our Hare and Tortoise class, we noticed that there are a lot of redundant codes. We will find the common things among hare and tortoise and design an Animal class. Afterwards we can derive any animal species from Animal class.

Recall the header files of Hare and Tortoise.

#ifndef \_HARE\_H

#define \_HARE\_H

class Hare

{

public:

Hare();

Hare(int\* pattern, int length, int position);

~Hare();

int getPosition() const;

void setPosition(int newPosition);

int\* getPattern() const;

int getPatternLength() const;

void move();

private:

int length;

int\* pattern;

int position;

};

#endif#ifndef \_TORTOISE\_H

#define \_TORTOISE\_H

class Tortoise

{

public:

Tortoise();

Tortoise(int\* pattern, int length, int position);

~Tortoise();

int getPosition() const;

void setPosition(int newPosition);

int\* getPattern() const;

int getPatternLength() const;

void move();

private:

int length;

int\* pattern;

int position;

};

#endif

From the above comparisons, we can see that both hare and tortoise have length, pattern (as an array of integers), and position information. So, an animal should have these data members (aka attributes) as well.

In addition, an animal should have information like name (a string) and a mark (a char). In previous design of Hare and Tortoise, we associate name and mark information directly with Hare and Tortoise.

In Animal Competition project, we do not know in advance which animal will participate in competition, so each animal should carry its information of name and mark. Also, a method to set pattern and its length should be provided since this setting cannot be put into constructors of Animal class. We declare this method as

void setPatternAndLength(int\* pattern, int size);

Set pattern and length together in the same method since array and its size should be a pair.

Furthermore, to illustrate what an animal will do after winning a race, we declare a method showExcitement. Different animal will show excitement in different ways, so this method should be overridden by subclasses of Animal. Here are some examples.

Duck shouts quack.

Hare jumps high.

Tortoise sticks out head.

Hence, we declare this method as virtual.

virtual void showExcitement() const;

In summary, the header file of Animal should look like as follows, where the highlight methods are not in Hare or Tortoise classes.

#ifndef ANIMAL\_H\_

#define ANIMAL\_H\_

#include <iostream> //std

class Animal

{

public:

Animal();

~Animal();

void setPatternAndLength(int\* pattern, int size);

void setMark(char mark);

void setName(std::string name);

void setPosition(int position);

int\* getPattern() const;

int getPatternLength() const;

int getPosition() const;

std::string getName() const;

char getMark() const;

void move();

virtual void showExcitement() const;

private:

int\* pattern;

int patternLength;

char mark;

std::string name;

int position;

};

#endif

**Motivation for polymorphism**: if our Competition class applies to all animals, then later on, whichever new species is generated, you do not need to change one line your Competition class.

You will reuse some of your codes in Hare and Tortoise Competition, but you need to modify some. So, remember to make a backup copy of your original source code.

1. Design class Animal, which includes data member position, move patterns, and mark id. In this application, for an animal, we care about the following attributes:
2. Species name of an animal, say hare, tortoise, duck, elephant.
3. current position of an animal
4. move patterns of an animal, represented in an array of integers, and
5. mark id of an animal. For example, the mark id for a hare is letter ‘H’; and mark id for a tortoise is letter ‘T’, and so on.
6. You need to write setters and getters for these data members.
7. Declare an abstract method showExcitement to print out a short phrase indicating what an animal will do to show its excitement when it wins.

For example, if a hare wins, it fluffs its tail; if a tortoise wins, it sticks out its head; if a duck wins, it quacks; if an elephant wins, it throws its nose.

virtual void showExcitement() const;

Keyword virtual before method showExcitement means the method is going to be overridden by subclasses of Animal. Keyword const after parameter list means showExcitement does not change the data members of the current object.

1. Extend Hare class from Animal. Write constructor(s) and set appropriate values for corresponding data members.
2. Similarly, extend Tortoise class from Animal.
3. Before you go to the next step, test your code to make sure that competition still works between a hare and a tortoise (whether you write Hare and Tortoise directly or derive these classes from Animal class, each animal still shares the same behavior.)
4. Next, we need to rewrite Competition class so that the competition will not be restricted to hare and tortoise only. This is an application of polymorphism. That is, the competition now involves a vector of animals and a road.

Here is the header file of Competition class.

#ifndef COMPETITION\_H\_

#define COMPETITION\_H\_

#include "Animal.hpp"

#include "Road.hpp"

#include <vector>

class Competition

{

public:

Competition();

~Competition();

void addRoad(int length); //add a road with those many blocks

void addPlayer(Animal\* beast);

void start();

private:

std::vector<Animal\*> players;

Road\* rd;

};

#endif

1. Since we do not know how many winners in the end, we use vector<Animal\*> to record the winners. Here is how a variable called winner is declared and instantiated.

vector<Animal\*> winners;

Note that each element of vector winners must be a pointer of Animal, otherwise, it would have "slicing" problem, that is, the subclass's info is sliced out, and only super class's info is kept. Remember the requirements of polymorphism?

1. In competition, you can use **push\_back** method of vector to add champion(s) of competition to vector winners.
2. Report the winner(s).
3. Note this project applies the concept of polymorphism: Competition class involves a vector of animals. In case we have a new species, we just need to extend that species from Animal class, then test the competition by adding one such animal to the animal array in CompetitionClient class. However, not one line of Competition needs to be modified.
4. Here is a sample run of my code (suppose hare, tortoise, elephant, and duck participate in a competition). Sometimes two animals are located in the same position, so only one mark is shown (see time stamp 12 of the following ouput)

start a game

1 D H

2 D T H

3 T H D

4 T H D

5 T D H

6 T D H

7 T D H

8 T D H

9 T DH

10 T D

11 HT D

12 H T D

13 H T D

14 H T D

15 H D T

16 H D T

17 H DT

18 H TD

19 H T D

20 H T D

21 H T D

22 H T D

23 H T D

24 H T D

25 H T D

26 H T D

27 H T D

28 H T D

winners

Duck shouts quack.

Sometimes there can be more than one winner, shown as follows, where I use a shorter road for competition.

start a game

1 D T

2 D HT

3 HT D

4 T D

winners

Hare jumps high.

Duck shouts quack.